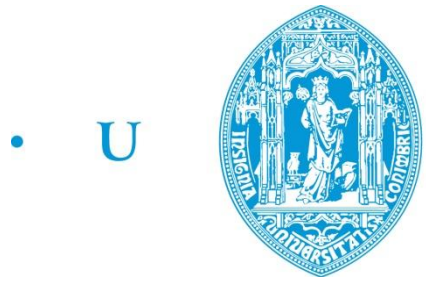


Universidade de Coimbra
Faculdade de Ciências e Tecnologia
Departamento de Engenharia Informática

Dificuldades de aprendizagem de programação de computadores: contributos para a sua compreensão e resolução

Anabela de Jesus Gomes

Dissertação submetida à Universidade de Coimbra para a obtenção do grau de "Doutor em Engenharia Informática", elaborada sob a orientação do Professor Doutor António José Mendes.

Coimbra

2010

Ao Álvaro

Ao João Bernardo

Agradecimentos

Ao Professor Doutor António José Mendes pela dedicação, disponibilidade, apoio constante e profissionalismo com que orientou todo o trabalho, desde a sua fase inicial, às valiosas críticas, ensinamentos e sugestões durante todo este percurso e em especial, pelas revisões do texto.

Aos professores Mestre Amâncio Santos, Doutora Ana Maria de Almeida, Doutora Ana Rosa Borges, Doutor Ernesto Costa e Mestre Maria Emília Bigotte os meus sinceros agradecimentos pela grande disponibilidade que demonstraram ao permitir a realização dos diversos estudos nas disciplinas que regiam, sem os quais este trabalho não teria sido possível.

A todos os professores que passaram nas suas turmas os instrumentos utilizados e às investigadoras Ana Pacheco, Joana Henriques e Scheila Martins.

Ao Instituto Superior de Engenharia de Coimbra pela autorização da realização de alguns estudos.

À Fundação para a Ciência e Tecnologia por todo o apoio financeiro proporcionado.

A toda a minha família não apenas pelo incentivo mas essencialmente por todo o tempo que lhe era devido.

Aos colegas do Instituto Superior de Engenharia de Coimbra que das mais variadas formas contribuiram para que a realização deste trabalho fosse possível.

Finalmente, os meus agradecimentos a todos aqueles que das mais variadas formas ofereceram contribuições para que o desenvolvimento deste trabalho fosse possível.

Resumo

As dificuldades existentes na aprendizagem da programação sugerem a existência de uma diversidade de défices em diferentes níveis, às quais os métodos de ensino e aprendizagem clássicos parecem não dar uma resposta suficientemente eficaz. A importância do problema já foi universalmente reconhecida, havendo diversas áreas e subáreas de investigação dedicadas ao assunto.

Neste trabalho investigámos alguns aspectos teóricos que nos pareceram mais pertinentes para o entendimento deste problema, nomeadamente questões relativas à capacidade de resolução de problemas, aos estilos de aprendizagem, bem como à utilização de elementos de apoio ao ensino de que se destacam a utilização de taxonomias de objectivos educacionais ou ferramentas informáticas de apoio ao ensino da programação.

Após algumas investigações que nos permitiram colocar determinadas conjecturas teóricas realizámos um conjunto de estudos de forma a fundamentá-las. Uma vez que o problema é complexo, utilizámos uma metodologia de triangulação, através da utilização de múltiplos métodos sobre uma série de estudos que ao longo do tempo foram acumulando alguma evidência. Ao juntar e comparar múltiplas fontes de dados umas com as outras e confrontando os métodos foi possível corroborar algumas das conjecturas inicialmente colocadas.

Terminamos o trabalho com um conjunto de recomendações e estratégias de ensino e aprendizagem que julgamos contribuir para atenuar o problema. Não é nossa convicção que a aplicação dessas propostas permita transformar qualquer aluno num programador brilhante. No entanto pensamos que a sua prossecução poderá fornecer uma boa estruturação para que um aluno regular, desde que fortemente motivado, possa compreender os conhecimentos básicos e aprender estratégias para realizar com sucesso as disciplinas introdutórias de programação.

Abstract

The difficulties found in programming learning suggest the existence of diverse flaws at different levels. The classic methods of education seem to be insufficient to deal with these difficulties. The importance of the problem was already universally recognized and has research areas and subareas devoted to the subject.

In this work we investigated some theoretical aspects that seemed more important for the comprehension of this problem. We investigated questions concerning problem solving abilities, learning styles, as well as the utilization of some learning support strategies and tools, namely the use of a taxonomy of educational objectives or software tools to support the programming teaching and learning.

After some research that allowed us to make some theoretical assumptions, we carried through a set of studies to back them up. As the problem is complex, we used a triangulation methodology, using multiple methods on a series of studies, which made possible to collect evidence during our research. Joining and comparing multiple sources of data and collating the methods it was possible to corroborate some of our initial assumptions.

We finished the work with the proposal of a set of teaching and learning recommendations and strategies that we think can contribute to minimize the problem. It is not our conviction that the application of these proposals will transform any student into an excellent programmer. However we believe that it is a good framework for any regular student that shows strong motivation. With these conditions we believe that these students can understand the basic concepts and develop strategies to carry through introductory programming courses successfully.

Índice

Lista de Figuras

Lista de Tabelas

Cap. 1

Introdução

"Nunca se sabe o que há para fazer, mas há sempre algo novo para se encontrar".

Merce Cunningham

1.1 Âmbito e Motivação

O ensino e aprendizagem de programação constituem um enorme desafio para alunos e professores. Os elevados níveis de insucesso nas disciplinas introdutórias de programação, em qualquer grau e sistema de ensino, em qualquer parte do mundo, são tema de preocupação e alvo de variadas pesquisas, ao longo dos tempos, resultando também em muitas propostas, sem que contudo tenham sido reportadas melhorias generalizadas.

O ensino da programação tem como propósito conseguir que os alunos desenvolvam as suas capacidades, adquirindo os conhecimentos e competências necessárias para conceber programas e sistemas computacionais capazes de resolver problemas reais. Bennedsen e Caspersen (2005b) referem que um dos objectivos mais importantes de uma disciplina introdutória de programação é que os alunos aprendam uma abordagem sistemática para desenvolver programas computacionais. 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 na compreensão e, em particular, na aplicação de noções básicas, como as estruturas de controlo, para a criação de algoritmos que resolvam problemas concretos. Estas dificuldades traduzem-se inevitavelmente em elevadas taxas de insucesso ou desistência.

A literatura inclui variadas descrições de casos de insucesso em disciplinas introdutórias de programação, por todo o mundo, independentemente da linguagem de programação usada. Apesar de se terem verificado enormes mudanças em termos tecnológicos, desde a invenção do computador digital, há panoramas que se têm mantido ou até agravado. Constata-se que um número significativo de alunos não consegue aprender a programar. Lister (2000) refere que em muitas universidades australianas, as taxas de insucesso em disciplinas introdutórias de programação se encontram entre as piores. Igualmente Bruce e McMahon (2002) referem as elevadas taxas de insucesso e a incapacidade dos alunos para completar pequenas tarefas em unidades introdutórias de programação. Estes autores referem também que a aprendizagem e ensino de programação no ensino superior constituem um eterno problema. Os professores estão familiarizados com alunos que se aproximam do seu projecto final de curso determinados a evitar a programação a todo o custo, presumivelmente porque não conseguem programar ou crêem que não o conseguem fazer (Carter e Jenkins, 1999). Dehnadi e Bornat (2006) referem também que existe uma enorme taxa de reprovação nas disciplinas introdutórias de programação nas universidades britânicas. Relatam ainda que entre 30% a 60% dos alunos de ciências da computação em cada universidade reprovam na primeira disciplina de programação. Estes autores destacam também que, apesar do esforço realizado por muitos professores e da quantidade de investigação referente a métodos de ensino, o panorama tem vindo a piorar ao longo dos anos, não fazendo ideia da causa deste problema.

Há evidências e vários estudos que revelam que a resolução de problemas de programação é uma tarefa difícil. Como referido por Denning (2004a):

"O grande desafio do ensino e aprendizagem de programação consiste em compreender o processo de programação e a prática do programador de forma a proporcional uma transferência educativa eficaz de conhecimentos e aptidões".

Winslow (1996) refere que aprender a programar é difícil e que os programadores novatos sofrem de uma variedade de dificuldades e défices. As disciplinas de programação são geralmente vistas como difíceis e têm frequentemente as mais elevadas taxas de desistência. Este problema tem sido constatado por diversos autores (Jenkins, 2002; Lahtinen et al., 2005; Dehnadi e Bornat, 2006; Lister et al., 2006; Simon et al., 2006).

Um estudo realizado em nove instituições de seis países teve como finalidade averiguar as competências de programação de estudantes universitários de ciências de computação, no final do seu 1º ano (McCracken et al., 2001). Os investigadores ficaram surpreendidos com os resultados que mostraram que após a realização das disciplinas de programação muitos alunos não sabem programar. Lister e seus colegas fizeram uma

pesquisa similar em sete países e chegaram à mesma conclusão (Lister et al., 2004). Dehnadi e Bornat (2006) referem também variados estudos para mostrar a universalidade deste problema.

Efopoulos e colegas concordam que a programação é uma disciplina cognitiva e inerentemente exigente, sugerindo ser a aptidão computacional mais difícil de dominar (Efopoulos et al., 2005). Diversos autores associam esta dificuldade à natureza abstracta da programação (Lahtinen et al., 2005; Bennedsen e Caspersen, 2006; Bennedsen, 2008). Por exemplo, Dunican (2002) adianta que noções como variáveis, tipos de dados, memória dinâmica, entre outros, não têm correspondência concreta no quotidiano e dominar estes conceitos fundamentais não é simples.

Diversos autores consideram que a aptidão para programar consiste não numa única capacidade, nem num conjunto de capacidades, mas antes envolve uma hierarquia de competências muitas das quais têm de estar activas simultaneamente (Sloane e Linn, 1988; Lister et al., 2004). Lister acrescenta ainda que quando um aluno é confrontado com uma hierarquia de exigências, geralmente aprende primeiro as de mais baixo nível e, depois, progride gradualmente no sentido das de nível superior. O caso da codificação (uma pequena parte das competências necessárias para programar) implica que os alunos aprendam a sintaxe básica e depois, gradualmente, progridam para a semântica, estrutura e finalmente estilo. Também Perkins et al. (1988) afirmam que a programação exige um esforço significativo envolvendo competências de diversas áreas. Outros autores, consideram que programar não envolve apenas mais do que uma única competência, mas envolve também mais do que um processo distinto (Jenkins, 2002). Por exemplo, este autor considera que existem múltiplos processos subjacentes que podem diferir em número consoante se trate de programadores experientes ou novatos. Em termos gerais, considera que no nível mais baixo a especificação tem de ser transformada num algoritmo o qual é então traduzido para um programa codificado. A parte mais difícil deste processo é a primeira. Dado o algoritmo correcto o outro processo é essencialmente mecânico e como tal os alunos já não apresentam grandes problemas. Verifica-se frequentemente a existência de alunos que conseguem acompanhar as matérias teóricas de programação, que conseguem dissecar e compreender programas, mas que são totalmente incapazes de escrever os seus próprios programas. Nestes casos, não dominaram todos os processos, pois podem conseguir fazer codificações mas não produzir um algoritmo.

Também Dijkstra (1989) refere que uma característica particular da programação é o facto de envolver resolução intensiva de problemas. Esta opinião é partilhada por outros autores que constatam este problema nas instituições de ensino dos seus países. Por exemplo, Dunican (2002) afirma que os actuais alunos irlandeses são o produto de sistemas

educativos primários e secundários que não têm módulos de resolução de problemas e lógica nas suas disciplinas. Este aspecto coloca os alunos Irlandeses em desvantagem quando comparados com os seus homólogos de outros países que aprendem este tipo de competências ao nível do ensino secundário e primário. Para Jenkins (2002) não há nada inerentemente difícil relativamente à programação, referindo que é simplesmente uma questão de competências, capacidades ou talentos e existem alunos que os não têm. As capacidades a que se refere são a resolução de problemas e a aptidão matemática. É também possível encontrar outros estudos que sugerem a existência de correlação entre as capacidades de programar e as competências matemáticas (Byrne e Lyons, 2001).

Outros autores mencionam outros problemas, por exemplo, Roberts (2004) como presidente da *Association for Computing Machinery (ACM) Education Board Task Force on Computer Science Education* comentou que o maior desafio relativamente à *Computer Science Education* (CSE) e que aumenta a complexidade com que os programadores novatos têm de lidar, refere-se à complexidade e instabilidade das linguagens de programação.

Outro factor frequentemente referido na literatura é que 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 (Ng e Bereiter, 1991). Já Ball (1977) referia que, para haver sucesso em qualquer tarefa, um indivíduo tem de estar fortemente motivado. Porém, também acontece frequentemente que os alunos demonstram interesse pela programação (Gayo-Avello e Fernandez-Cuervo, 2003), mas a maioria deles acha-a difícil e uma tarefa cognitiva complexa (Simon, 1973; Jeffries et al., 1980; Mayer et al., 1986; Kim e Lerch, 1997), pelo que a motivação, só por si, não será suficiente para enfrentar o problema.

Apesar de menos usual também é possível encontrar referências alusivas ao facto de as disciplinas de programação estarem mal localizadas no currículo, num momento de muitas dificuldades e novidades de uma vida nova e autónoma para o aluno. A programação é, normalmente, ensinada como um assunto básico no início de um curso superior, coincidindo com um período da transição e instabilidade na vida do aluno. Jenkins (2002) refere que o tipo de assunto é já suficientemente difícil quando os alunos estão estáveis, quando colocado num período de transição pode contribuir para aumentar ainda mais esta dificuldade.

Consideramos de extrema relevância as causas anteriormente mencionadas, referidas na literatura. Desta forma, a motivação principal desta tese consiste na investigação, de um conjunto de factores relativos às causas de insucesso a programação e consequente proposta de soluções, com o intuito de contribuir para minorar este problema. Porém, para melhor enquadrar as possibilidades de exploração da problemática referente ao ensino e aprendizagem de programação, é primeiramente apresentada a área de CSE em que esta temática se insere. Neste sentido, pretende-se, proporcionar ao leitor uma perspectiva geral referente à investigação nesta área, estruturada de acordo com as suas subáreas de pesquisa principais.

1.2 Investigação sobre Educação em Ciências da Computação

A investigação em educação (ensino e aprendizagem) em ciências da computação abrange actualmente uma grande variedade de assuntos, sem que por vezes haja consenso sobre o enquadramento de determinada temática. Contudo, consideramos crucial o contributo do livro *Computer Science Education Research*, editado por Sally Fincher e Marian Petre, para a estruturação da área (Fincher e Petre, 2004). Neste, as autoras identificaram dez grandes áreas referentes à educação em ciências da computação. Apesar de as áreas não serem disjuntas, esta classificação poderá constituir uma útil ferramenta para um melhor posicionamento aquando da investigação de determinado tópico referente ao ensino/aprendizagem de ciências da computação. As dez áreas mencionadas são:

1. Compreensão do aluno;
2. Animação, visualização e simulação;
3. Métodos de ensino;
4. Avaliação;
5. Tecnologia educacional;
6. Transferência de práticas profissionais para a sala de aula;
7. Inclusão de novos desenvolvimentos e tecnologias;
8. Transferência do ensino presencial para o ensino à distância;
9. Recrutamento e retenção;

10. Construção da disciplina.

Será apresentada uma breve descrição de cada área relacionando-a com o ensino e aprendizagem da programação em geral e o interesse evidenciado para esta tese em particular. A cada uma delas será dada maior ou menor ênfase consoante o interesse que tem para este estudo particular.

1.2.1 Compreensão do aluno

A investigação realizada nesta área incide principalmente no estudo sobre os modelos mentais e conceptuais que os alunos têm sobre determinado assunto bem como as suas concepções e *misconceptions*[1] sobre os mesmos. Muitos dos estudos realizados nesta área objectivam compreender a razão pela qual os estudantes têm problemas com um determinado tópico, conceito ou construção particular. Também neste âmbito se encontram variados estudos referentes às competências, comportamentos e atitudes que distinguem os bons alunos dos alunos com fracos desempenhos. As diferenças em termos de compreensão e percepção dos assuntos entre estudantes principiantes e peritos são, consequentemente, alvos de estudo nesta área. A gama de tópicos investigada é também vasta, podendo incluir temáticas mais alargadas como "Que competências de planeamento e concepção de programas é que os alunos apresentam?" ou "Como é que os alunos aprendem determinados paradigmas de programação?" a questões mais específicas do tipo "Como é que os alunos aprendem a recursividade?". A investigação existente nesta área não é específica das ciências da computação, mas antes concebida como uma pesquisa multidisciplinar integrando frequentemente aspectos das ciência e psicologia cognitivas. Com consideráveis desenvolvimentos nesta área destaca-se o grupo de interesse em psicologia da programação PPIG (*Psychology of Programming Interest Group*)[2].

Dentro desta área, consideramos de interesse as subáreas a que o grupo supracitado se dedica, organizadas em cinco categorias: estudos psicológicos, concepções e *misconceptions*, estudos fenomenográficos, comportamento dos principiantes em ciências da computação e compreensão do educador. De seguida é feita uma pequena explanação de cada uma destas categorias. De todas elas, dentro desta área, a mais relevante para a temática estudada é a designada por "estudos psicológicos", pelo que também será a mais descrita.

[1] Neste documento utiliza-se este estrangeirismo, por não se ter encontrado uma tradução considerada adequada.
[2] Disponível em http://www.ppig.org. Estamos gratos à Professora Doutora Maria José Marcelino pela descoberta deste grupo de investigação.

1.2.1.1 Estudos psicológicos

Os estudos psicológicos da programação tiveram o seu início nos anos 70 do século XX, porém, como referenciado por Sheil (1981), muitos desses estudos eram metodologicamente fracos, melhorando, posteriormente, com a entrada de psicólogos nesta área de investigação. Nos anos 80, motivados pelo crescimento da ciência cognitiva, diversos investigadores conduziram pesquisas referentes ao desempenho de programadores principiantes. Muito do trabalho centrou-se na identificação dos problemas que os alunos enfrentavam ao aprender a programar. Estudos conduzidos por diversos autores, de que se destacam Soloway e Spohrer, concluíram que os programadores principiantes podem saber a sintaxe e a semântica das instruções individuais, mas não sabem como combiná-las de forma a constituir programas válidos (Soloway et al., 1983; Soloway, 1986; Spohrer e Soloway, 1986; Soloway e Spohrer, 1988). Spohrer e Soloway (1986) fazem a distinção entre dificuldades baseadas na composição (dificuldades em juntar todas as peças) e dificuldades baseadas na construção (concepções erradas sobre as estruturas da linguagem). Os estudos que realizaram demonstraram que as dificuldades dos alunos são devidas aos problemas baseados na composição e não na construção. Aqueles autores referiram que os educadores podem contribuir para melhorar o desempenho dos seus estudantes ensinando-lhes estratégias de composição. Este resultado é coerente com estudos anteriores realizados por outros investigadores (Mayer, 1981; Linn e Dalbey, 1985) que analisaram a distinção entre a capacidade de interpretar programas (compreensão de programas) e a capacidade de escrever ou compor programas (geração de programas). As descobertas de Spohrer e Soloway foram confirmadas por pesquisas similares realizadas nos anos 90. Segundo estas, a capacidade de resolver um problema exige aptidões que vão além da sintaxe e semântica de uma linguagem de programação e os erros existentes nos programas dos alunos não estão geralmente relacionados com a sintaxe mas com deficientes estratégias de resolução de problemas e insuficiente planeamento (Scholtz e Wiedenbeck, 1992; Shackelford e Badre, 1993; Wiedenbeck et al., 1993; Anjaneyulu, 1994). Winslow (1996) refere que os alunos podem saber a sintaxe e a semântica de declarações individuais, mas não sabem como combiná-las de forma a gerar programas válidos. O mesmo autor concluiu que há uma correspondência muito pequena entre a capacidade de escrever um programa e a capacidade de o interpretar. Estudos mais recentes confirmam a persistência destes problemas. Segundo os autores, a explicação para tal incapacidade é a falta de conhecimentos e competências de resolução de problemas que os alunos apresentam. Ou seja, os alunos não conseguem, a partir da descrição de um problema decompô-lo em subproblemas, implementá-los e posteriormente juntar todas as subsoluções numa solução completa. Tentando refinar esta tarefa, o grupo de trabalho "ITiCSE 2004" estudou uma explicação alternativa, defendendo que muitos alunos apresentam um domínio frágil quer dos princípios básicos de

programação quer da capacidade de sistematicamente realizar tarefas de programação rotineiras tais como interpretar código (Lister et al., 2004). Uma continuação do estudo referente à compreensão de programas é documentada em Lister et al. (2006) usando a taxonomia SOLO (Biggs e Collis, 1982). Neste artigo os autores referem o uso desta taxonomia para testar o conhecimento de programação de programadores inexperientes. Os autores defendem que os professores devem usar estratégias de avaliação adequadas, testando os alunos em todos os níveis da taxonomia. Um aluno não deverá apenas conseguir interpretar linhas de código isoladas mas saber integrá-las numa estrutura coerente e usar essa estrutura para resolver a tarefa:

"…o aluno deve estar preparado para ver a floresta e não apenas as árvores…"

Segundo aqueles autores, falta frequentemente aos alunos a capacidade de interpretar código e descrevê-lo em termos relacionais, pelo que, nessas situações, não estarão intelectualmente bem equipados para escrever código similar. Outros estudos concluíram que os estudantes não conseguiam gerar programas, compreendê-los ou projectá-los a níveis aceitáveis (Mead et al., 2006). Robins et al. (2003) fornecem uma revisão detalhada da literatura em relação ao estudo psicológico/educacional da programação, chegando à seguinte conclusão:

"A recomendação principal que emerge da literatura é a de que o ensino se deve centrar não somente na aprendizagem das características de determinada linguagem de programação, mas igualmente na combinação daquelas características e especialmente no problema subjacente referente ao projecto de programas básicos. [...] sugerimos que as estratégias de programação devam receber uma atenção cada vez mais explícita em disciplinas introdutórias de programação. Uma forma de o conseguir poderá passar por introduzir muitos exemplos à medida que os programas são desenvolvidos, discutindo as estratégias usadas como parte deste processo" (Robins et al., 2003).

Interessante, mas também decepcionante, foi o facto de Soloway ter sugerido essencialmente o mesmo há mais de vinte anos (Soloway, 1986). Em conclusão, passado todo este tempo, estudo após estudo, multi-institutional ou multinacional, evidencia-se que muitos alunos não conseguem aprender a programar e que os problemas graves que experimentam são baseados na composição - como juntar todas as peças.

1.2.1.2 Concepções e *misconceptions*

Identificar *misconceptions* e as suas causas, constitui uma área significativa de pesquisa, existindo literatura considerável que refere como as *misconceptions* complicam a aprendizagem (CUSE, 1997). Outra perspectiva interessante é fornecida por Smith e colegas

no artigo "*Misconceptions Reconceived*" (Smith et al., 1993). Segundo a abordagem tradicional as *misconceptions* são equívocos que impedem a aprendizagem. Porém, na óptica destes autores as *misconceptions* não devem ser vistas tão negativamente, mas antes encaradas como elementos chave da aprendizagem. Os autores referem que as concepções que conduzem a conclusões erradas num determinado contexto podem até ser muito úteis noutros contextos. Clancy fornece também uma perspectiva geral sobre as *misconceptions* explicando como se formam e fazendo referência a variados exemplos que revelam que a maioria das *misconceptions* relacionadas com programação são devidas a transferências impróprias e confusões sobre modelos computacionais (Clancy, 2004). Este autor refere que, na maioria dos casos a correspondência entre os erros dos alunos e a *misconception* subjacente é clara, mas que existem diversas outras circunstâncias onde essa conexão não é tão óbvia. Aqueles autores referem também que cada paradigma de programação ou nova tecnologia tem as suas *misconceptions* associadas, as quais os professores devem conhecer bem para ajudar os alunos a aprender eficazmente. Aqueles autores descrevem também várias situações que revelam atitudes impróprias de programação, discutem maneiras de tratar as *misconceptions* e sugerem direcções para a pesquisa nesta área.

1.2.1.3 Estudos fenomenográficos

Os estudos fenomenográficos consistem numa abordagem que se centra nas diferentes formas pelas quais os indivíduos experienciam, percebem e compreendem um determinado fenómeno (Marton e Booth, 1997; Lister, 2003). No princípio dos anos 70, os estudos fenomenográficos identificaram duas aproximações diferentes utilizadas pelos estudantes nas suas aprendizagens, nomeadamente a aprendizagem profunda e a superficial. Na abordagem à aprendizagem "profunda", os estudantes tentam desenvolver uma compreensão genuína do que estão a estudar enquanto que os estudantes que utilizam uma abordagem "superficial" têm como único objectivo completar as tarefas propostas pelo professor, com um esforço mínimo. A pesquisa fenomenográfica inspirou o desenvolvimento de práticas de ensino incentivadoras de uma aprendizagem profunda (Biggs, 2003). Nas ciências da computação, e especificamente na temática referente à aprendizagem de programação, Booth conduziu um relevante trabalho de investigação fenomenográfico estudando as diferentes formas pelas quais os estudantes experienciam a programação e mais especificamente a recursividade (Booth, 1997).

Mais recentemente, a fenomenografia foi aplicada para perceber as diferentes formas pelas quais os estudantes inexperientes percebem os conceitos básicos em programação orientada a objectos (Bruce et al., 2004; Eckerdal e Thuné, 2005; Eckerdal et al., 2005; Stamouli e Huggard, 2006). Também Lindholm tem conduzido estudos fenomenográficos que visam

compreender as diferentes perspectivas que estudantes de humanidades e de ciências da computação têm sobre os conceitos de classes, objectos e outros aspectos inerentes a aspectos introdutórios de programação orientada a objectos (Lindholm, 2005; Lindholm, 2007).

1.2.1.4 Comportamento do aluno

Os estudos empíricos referentes ao comportamento de programadores inexperientes recaem tipicamente em análises do código por eles desenvolvido ou de respostas suas em testes. Apesar desse tipo de dados fornecer alguma informação referente ao tipo de programação realizada, dizem pouco sobre os processos de programação em que os indivíduos se envolvem. Hundhausen et al. (2006) reuniram um conjunto de vídeos mostrando alunos enquanto codificavam. A metodologia desenvolvida permitia aos investigadores verificar a parte do código onde o aluno estava focalizado, verificando se os participantes utilizavam o seu tempo concentrados em actividades de programação produtivas, se eram capazes de encontrar e corrigir erros semânticos no seu código, se validavam o código, bem como o tempo dedicado a essas tarefas e se o *feedback* gerado ajudava ou impedia os participantes de prosseguirem nas suas actividades (Hundhausen et al., 2006). Apesar de se desconhecer a existência de estudos que relacionem o comportamento dos alunos enquanto programam com o seu desempenho final, existem alguns indicadores. Pode-se citar, por exemplo, o trabalho de Jadud que explorou o que chama de comportamento de compilação de principiantes, isto é, o comportamento que um aluno inexperiente em programação adopta enquanto repetidamente edita e compila programas (Jadud, 2006). Jadud define o chamado quociente de erro (EQ) que caracteriza o quanto um estudante se esforça relativamente aos erros de compilação. Apesar de ter encontrado uma certa correlação entre o EQ e o desempenho do aluno a programação, Jadud conclui que não pode fazer nenhuma constatação forte sobre se o comportamento que um aluno demonstra ao compilar um programa pode ser usado como um previsor para o seu desempenho medido através de um exame tradicional.

1.2.1.5 Compreensão do educador

Embora este tópico saia fora do âmbito deste trabalho de investigação, é interessante notar que alguns investigadores têm recentemente aplicado alguns dos estudos feitos com estudantes aos professores. Destaque-se os trabalhos de Bennedsen e Schulte sobre a percepção dos professores relativamente à utilização do paradigma de orientação a objectos como primeira abordagem de programação (Bennedsen e Schulte, 2007). Relevante é também o estudo fenomenográfico conduzido por Thompson referente às percepções sobre o desenvolvimento de programas utilizando paradigmas de orientação a objectos, de académicos que ensinam esta temática e profissionais experientes na área (Thompson, 2006).

Os estudos encontrados colocam os professores e profissionais na posição de peritos e mostram os aspectos em que as suas metodologias e procedimentos para resolver problemas diferem dos principiantes. Surgem no entanto, na literatura, recomendações sobre o tipo de professores que deverá ensinar programação. Sobre este aspecto as advertências recaem sempre sobre os professores mais experientes e não sobre os principiantes. Constata-se porém, em muitas instituições nossas conhecidas que se passa exactamente o contrário, havendo inclusivamente a crença de que qualquer pessoa com habilitações informáticas está habilitada a ensinar programação.

1.2.2 Animação, visualização e simulação

Existe uma grande parte de investigadores em CSE que, como informáticos que normalmente são, se dedicam à construção de sistemas, programas computacionais, com o intuito de melhorar a aprendizagem do aluno. Apesar de existir uma grande variedade deste tipo de ferramentas e diversas tentativas para as classificar, sendo a mais completa conhecida a apresentada em Kelleher e Pausch (2005), a generalidade baseia-se na animação (muitas vezes de algoritmos específicos), visualização e simulação. A motivação subjacente a esta área é a de apelar ao potencial do sistema visual humano. Assim, uma vez que os programas computacionais podem ser pouco claros quando apresentados num formato textual, é esperado que o formato gráfico animado contribua para uma melhor compreensão. A animação e visualização de algoritmos e de estruturas de dados dominam a área (Naps et al., 2003).

A partir dos anos 90, nota-se que a atenção de muitos investigadores foi dirigida para a eficácia pedagógica da animação e visualização de algumas das ferramentas desenvolvidas. Cite-se por exemplo, os estudos realizados por Hundhausen e colegas referentes à eficácia das visualizações de algoritmos. Nestes trabalhos, apresentaram uma revisão abrangente de 24 estudos de experiências controladas, incluindo uma classificação e análise respeitante às teorias de aprendizagem subjacentes às diversas visualizações (Hundhausen et al., 2002). A conclusão geral do estudo é a de que a forma como os estudantes usam essas ferramentas parece ter um grande impacto e eficácia educacional, permitindo que os alunos se envolvam activamente no processo de aprendizagem. Estes autores acrescentam que os alunos utilizam constantes reflexões do tipo "o que acontece se" nas tarefas de análise do comportamento algorítmico, envolvendo-se frequentemente em exercícios de previsão, pelo que estas ferramentas funcionam essencialmente como um catalisador da aprendizagem. O efeito positivo da utilização de visualizações de programas foi também documentado recentemente (Ebel e Ben-Ari, 2006).

1.2.3 Métodos de ensino

Existem vários aspectos que motivam os investigadores nesta área. Um deles diz respeito à forma como os educadores podem "construir pontes" para os alunos, proporcionando uma melhor aprendizagem. O trabalho de Sopher e seus colegas (Sopher et al., 1985) sobre planos de programação constitui um bom exemplo. Também o trabalho de estudos de caso conduzido por Linn e Clancy teve um grande impacto nesta área (Linn e Clancy, 1992). Esses estudos mostram a aplicação de importantes princípios para a concepção de programas, fornecendo explicações sobre como podem ser aplicados na realização de programas razoavelmente complexos, de grande utilidade para ajudar os estudantes a desenvolver importantes capacidades de programação.

A utilização de padrões para ajudar a desenvolver competências de programação, desde as mais básicas às mais avançadas constitui outro exemplo de esforços realizados nesta área, como documentado em Astrachan et al. (1998).

Existe também algum trabalho motivado por descobertas noutras áreas, de que se destacam a psicologia e sociologia, relativo a questões como a "aprendizagem activa", "estilos cognitivos", "estilos de aprendizagem", entre outros.

Esta é uma área muito extensa e que Caspersen divide em seis subáreas: ensino centrado no aprendiz/aluno, construtivismo, estudos de caso e aprendizagem, currículo invertido, padrões e ensino/aprendizagem de conceitos orientados a objectos (Caspersen, 2007). Dada a relevância do trabalho recente deste autor, faremos de seguida uma breve incursão por algumas das subáreas referidas que consideramos relevantes para a temática estudada.

1.2.3.1 Ensino centrado no aprendiz e construtivismo

Numa edição especial das comunicações da *Association for Computing Machinery*, Norman e Spohrer escreveram um pequeno editorial, que começa da seguinte forma:

"Está a ocorrer uma revolução na educação, que lida com a filosofia de como se ensina, do relacionamento entre professor e aluno, da forma como a sala de aula está estruturada e da natureza do currículo" (Norman e Spohrer, 1996).

As questões básicas podem ser descritas com palavras-chave como construtivismo, *learner-centered* e *problem-based*. A ideia fundamental é a de que as pessoas aprendem melhor quando estão imersas no tópico a aprender, motivadas para procurar novos conhecimentos e competências, por precisarem delas para resolverem o problema em questão. O objectivo é a exploração, construção e aprendizagem activas em vez da passividade de assistir a aulas

expositivas. Esta aprendizagem centrada no aluno é um tanto parecida com a abordagem do projecto de interfaces modernas, centrado no utilizador. O foco reside nas necessidades, competências e interesses dos aprendizes e na utilização de problemas autênticos e atraentes para o aluno. No artigo "*How much choice is too much?"* (Becker, 2006), o autor escreve:

"Fornecer uma perspectiva centrada no aprendiz está de acordo com aproximações construtivistas modernas da aprendizagem, significando que os cursos devem ser projectados com os atributos e gostos do aprendiz em mente".

De notar que, a adequação das actividades às preferências dos alunos, incluindo a liberdade de escolha e de flexibilidade dada ao aluno para a construção da sua aprendizagem, não contradiz a sua responsabilização, mas antes poderá contribuir para uma aprendizagem auto-regulada. Becker delineia numerosas estratégias para proporcionar escolha e flexibilidade aos estudantes num curso de programação. Os estratagemas incluem a escolha do problema a resolver, prazos flexíveis, a possibilidade de submeter novamente um trabalho que já tenha sido avaliado, contribuição do aluno para o conteúdo do curso, bónus para aperfeiçoamentos e trabalhos extra, entre outros. Acerca do mesmo assunto, Bergin e colegas relatam um estudo sobre a investigação do efeito da aprendizagem auto-regulada no desempenho em programação introdutória (Bergin, S. et al., 2005). Os resultados do estudo indicam que a aprendizagem auto-regulada é importante para aprender a programar e pode ser usada para prever parcialmente o desempenho do aluno na disciplina.

Diversos investigadores têm dirigido a questão do ensino centrado no aluno para a noção de construtivismo, segundo a qual é o aluno que deve construir activamente o seu conhecimento, a partir das reflexões das suas experiências, em vez de meramente o receber e armazenar, pela transmissão do professor:

"Aprender ocorre com o comportamento activo do estudante: é o que ele faz que aprende, não o que o professor faz" (Tyler, 1949).

Em geral, consideram-se duas escolas do construtivismo: a escola do construtivismo individual (ou cognitivo) de Piaget (Forman e Fosnot, 1982) e a escola do construtivismo social de Vygotsky (Vygotsky, 1978). As teorias construtivistas focam ora o aspecto individual ora o aspecto social da construção do conhecimento. O construtivismo individual de Piaget entende o conhecimento como sendo construído individualmente, dentro de um contexto, por meio da reflexão. De acordo com Piaget existem quatro estágios de desenvolvimento onde, individualmente, se constroem estruturas cognitivas. Piaget considera que, embora a interacção social seja importante, não é factor determinante para a mudança de pensamento do indivíduo. Vigotsky defende um modelo de aprendizagem de cognição social onde a cultura social e as actividades colaborativas constituem um

determinante principal do desenvolvimento e aprendizagem (Woolfolk et al., 2006). Ben-Ari analisou o construtivismo no contexto da CSE (Ben-Ari, 2001). O autor refere que dada a importância crucial da teoria da aprendizagem construtivista e a sua influência na pedagogia, os educadores de ciências da computação devem estudar esta teoria, realizar pesquisa e analisar as suas propostas educativas em termos do construtivismo. Também os indivíduos que desenvolvem programas e linguagens de programação devem ser guiados pelos princípios construtivistas.

1.2.3.2 Estudos de caso e aprendizagem

Ao longo dos anos, têm sido feitas várias tentativas no sentido de encontrar formas mais eficazes para motivar os alunos no que respeita ao que se pretende ensinar. No caso das disciplinas introdutórias de programação podemos destacar como crucial a ideia defendida em Pattis (1990) e o trabalho de Linn e Clancy (1992). No primeiro caso, a filosofia aplicada para o ensino da programação consistia em os alunos desenvolverem projectos concretos e reais (com simplificações adequadas), através de um método de desenvolvimento gradual. Na segunda situação, os autores desenvolveram determinado número de estudos de caso, cada um dos quais incluía a) a descrição do problema de programação; b) uma descrição do processo usado por um perito para resolver o problema, escrito de forma perceptível pelo estudante; c) uma listagem do código do perito; d) questões de estudo que permitiam praticar a concepção/planeamento, resolução e análise de programas e e) questões para avaliar a compreensão dos alunos relativa à solução apresentada.

Igualmente significativo foi o trabalho de Astrachan e seus colegas ao utilizarem nas disciplinas introdutórias de programação uma abordagem de aprendizagem aplicada (Astrachan e Reed, 1995; Astrachan et al., 1997). Nesta abordagem os alunos eram incentivados a interpretar, estudar, modificar e ampliar os programas escritos por programadores experientes e peritos, em contextos que utilizavam exemplos reais. Barnes e Kölling escreveram um livro (Barnes e Kölling, 2006) de acordo com estes princípios pedagógicos, seguindo uma aproximação orientada por problemas e centrada no aprendiz, onde as tarefas particulares de programação é que motivavam a aprendizagem das estruturas de programação necessárias e não o contrário.

1.2.3.3 Currículo invertido

Uma proposta profunda pelo menos considerando o momento em que foi apresentada consistia em adiar o ensino de factos até serem realmente necessários. Reek (1995) sugere uma aproximação para ensinar programação introdutória focando-se na compreensão das abstracções representadas pelas estruturas de dados clássicas sem considerar a sua

implementação física. Somente depois de os estudantes estarem confortáveis com o comportamento e aplicações das principais estruturas de dados é que aprendiam a implementá-las. A proposta de Reek é uma encarnação da ideia de Meyer do currículo invertido (Meyer, 1993). Reek não foi o único a implementar esta aproximação, Decker e Hirshfield (1993) descrevem uma abordagem similar. Concretizações mais recentes desta ideia são documentadas por Roumani (2006) e Pedroni e Meyer (2006).

Buck e Stucki (2000) referem que as abordagens tradicionais utilizadas nas disciplinas introdutórias de programação não estão em conformidade com a teoria da aprendizagem cognitiva. Acrescentam ainda que essas abordagens utilizam uma ordem de aprendizagem inversa àquela proposta pela taxonomia dos objectivos educacionais de Bloom (Bloom et al., 1956). O título do artigo é "*Design early considered harmful: Graduated exposure to complexity and structure based on levels of cognitive development*" e a mensagem é a de que a ordem que melhor condiz com a hierarquia do desenvolvimento cognitivo de Bloom é o reverso da ordem das actividades usadas no modelo clássico do ciclo de vida de *software*. A abordagem proposta por estes autores é caracterizada como "o ensino de desenvolvimento de *software* de dentro para fora em vez de começar com aplicações de consola ou projectos monolíticos".

1.2.3.4 Padrões

A motivação fundamental para uma abordagem do ensino da programação baseada em padrões reside na crença de que os padrões capturam pedaços do conhecimento. De acordo com a ciência cognitiva e a psicologia educacional, o ensino explícito de padrões reforça a aquisição de esquemas desde que a carga cognitiva total seja controlada. Ao longo dos últimos dez anos, diversos educadores de ciências da computação começaram a incorporar padrões de programação nos seus cursos não graduados. Refira-se como exemplo Wallingford (2000). Ideias similares aos padrões foram desenvolvidas por Mayer (1981). Convicto de que ensinar a sintaxe e a semântica de uma linguagem não é suficiente em disciplinas introdutórias de programação, este autor propõe uma estratégia de ensino, segundo a qual devem ser dadas aos alunos directrizes explícitas, similares às que as pessoas usam para resolver os problemas no seu quotidiano. Nomeadamente, mecanismos, explicações, objectivos, planos, regras de programação, métodos de composição de planos, entre outros. Também Soloway (1986) e Rist (1989) destacam a importância dos planos de programação, bem como a forma como planos mais simples poderão integrar outros mais complexos na resolução de problemas de programação. A consideração de padrões acelerou a aparição de um livro pioneiro neste tópico, "Design Patterns" (Gamma et al., 1995). Pode-se citar também um importante trabalho nesta temática realizado por um grupo que desenvolveu um catálogo dos chamados padrões elementares destinados a principiantes de

programação (Bergin, 2006a). A pertinência deste tópico pode ser averiguada pela recente edição especial sobre padrões pedagógicos da *Computer Science Education* (Fincher, 2006).

1.2.4 Avaliação

Esta área pode ser dividida em três subáreas: métodos/tipos de avaliação, avaliação automatizada e validade da avaliação. Algumas das questões tratadas nestas áreas dizem respeito aos diferentes tipos de avaliação, tentando compreender quais os que são mais adequados para determinados objectivos ou contextos particulares, bem como o que os torna eficazes. Nesta área são também realizadas investigações referentes a sistemas de avaliação automática de programas. Por exemplo, Ala-Mutka discutiu um grande número dessas ferramentas, detalhando os pontos fortes e fracos de cada uma delas (Ala-Mutka, 2005). Outras investigações realizadas nesta área objectivam compreender a validade de determinada avaliação, ou seja, se determinada avaliação afere realmente o tipo de conhecimentos ou competências que o educador pretende avaliar (Fincher e Petre, 2004). Outras questões alvo de investigação nesta área dizem respeito à verificação da autenticidade dos exames e trabalhos realizados pelos alunos, bem como a questões de plágio (Dick et al., 2003; Lancaster e Culwin, 2004).

1.2.4.1 Métodos de avaliação

Existem variados métodos de avaliação. Sittings (2005) enumera quatro categorias principais: resposta seleccionada (por exemplo, perguntas de escolha múltipla ou questões de resposta breve), ensaio/experiência (por exemplo, apresentação de um poster ou de um relatório escrito), avaliação de desempenho (por exemplo, estudo de caso, projecto ou jornal/diário reflexivos) e comunicação pessoal (por exemplo, apresentação de uma aula, entrevista ou contrato de aprendizagem). De acordo com este autor, cada categoria apresenta diferentes vantagens no que concerne à avaliação de determinados resultados de aprendizagem.

Os testes de escolha múltipla têm também, recentemente, sido reconhecidos como uma útil ferramenta de avaliação, a nível do ensino superior (Brown, 2001; Biggs, 2003; Woodford e Bancroft, 2005; Roberts, 2006), em contraste com perspectivas anteriores que suportam a ideia de que este tipo de avaliação serve apenas para avaliação de aprendizagens superficiais. As vantagens deste tipo de testes são muitas, nomeadamente a objectividade na correcção para além de permitirem avaliar, quando bem concebidos, conhecimentos mais profundos e não superficiais como geralmente se supõe. Em termos de avaliação de competências de programação não é comum os professores enveredarem por esta via de avaliação, optando, frequentemente, por exames escritos que implicam a realização de

programas completos e/ou trabalhos práticos/laboratoriais. Porém, têm sido apresentadas diversas propostas no sentido de facilmente avaliar competências de diferentes níveis de programação, utilizando testes de escolha múltipla. Veja-se a título de exemplo as recomendações dadas por Traynor e Gibson (2005) para a concepção de testes eficazes para esta finalidade. Existe também literatura dedicada à construção e análise desse tipo de testes (Ebel e Frisbie, 1986; Linn e Gronlund, 1995; Haladyna, 1999).

A avaliação pode também endereçar diferentes tipos de aprendizagens e consequentemente diferentes tipos de tarefas ou questões. Lister e Leaney propõem tarefas e actividades de avaliação de natureza diferente reflectindo os diferentes níveis da taxonomia de Bloom, aplicados a uma disciplina de programação (Lister e Leaney, 2003b). A ideia destes autores é a da utilização de uma aproximação em que são dadas diferentes tarefas aos alunos de acordo com os seus conhecimentos e capacidades, reflectindo, cada tipo de tarefa, os diferentes níveis da taxonomia de Bloom.

1.2.4.2 Avaliação automatizada

Há actualmente diversos sistemas para avaliação de diferentes aspectos relativos à programação realizada pelos alunos. Existem sistemas que testam as respostas dos alunos face a um conjunto de dados de entrada, outros avaliam automaticamente a funcionalidade de pequenas entidades, como funções ou métodos, em vez de programas completos. Outros testam programas com interfaces gráficas ou ainda programas relativamente ao estilo da codificação utilizada.

Apesar de existirem alguns sistemas para avaliação automatizada, apenas recentemente, começaram a surgir alguns trabalhos referentes a apreciações desses sistemas (Korhonen et al., 2002; Kumar, 2005a; Malmi et al., 2005b; Karavirta et al., 2006, Traynor et al., 2006), sendo também necessários estudos que façam a comparação entre eles. A avaliação automatizada é coberta por duas revisões recentes. Ala-Mutka (2005) fornece um levantamento de abordagens de avaliação automatizadas para tarefas de programação e o *Journal of Educational Resources in Computing* publicou recentemente uma edição especial sobre avaliação automatizada de tarefas de programação, prefaciada por Brusilovsky e Higgins (2005). Desta edição especial, destaca-se em particular o artigo "*Automatic test-based assessment of programming: A review"* (Douce et al., 2005).

O plágio é também um tema importante no ensino de ciências da computação como exposto por um grupo de trabalho do "ITiCSE 2002" (Dick et al., 2003). Lancaster e Culwin (2004) fornecem uma comparação de mecanismos para detecção de plágio em código fonte.

Daly e Horgan (2005) apresentam um sistema baseado em marcas de água, permitindo distinguir o fornecedor e o receptor, no caso de plágio.

1.2.4.3 Validade da avaliação

De considerável importância é também a investigação dedicada à compreensão sobre a validade da avaliação, isto é, se determinada avaliação verifica o tipo do conhecimento que o professor realmente quer avaliar (Fincher e Petre, 2004). Outro aspecto refere-se à avaliação das competências de programação. Estudos realizados por McCracken et al. (2001) revelam que muitos estudantes de computação não conseguem desenvolver programas correctos após uma disciplina introdutória de programação ou mesmo após a obtenção de classificações positivas a algumas disciplinas de programação. Daly e Waldron (2004) referem que na avaliação de um estudante deve ser dada ênfase à questão relativa às reais competências de programação adquiridas. No mesmo trabalho, estes autores examinam ainda a razão pela qual os métodos tradicionais de avaliação (exames escritos e trabalhos de programação) são falaciosos e defeituosos, considerando um outro método de avaliação (avaliações laboratoriais) como mais correcto. Os autores explicam também a razão da necessidade de uma avaliação rigorosa, a fim de incentivar os estudantes a desenvolverem reais capacidades de programação.

1.2.5 Tecnologia educacional e inclusão de novos desenvolvimentos e tecnologias

Neste âmbito têm sido desenvolvidos diferentes tipos de ferramentas, nomeadamente ambientes integrados de desenvolvimento (IDEs), micromundos de aprendizagem, sistemas de animação de algoritmos, entre outras ferramentas dos mais variados tipos. Kelleher e Pausch (2005) fizeram uma extensa classificação desse tipo de ferramentas, desenvolvidas ao longo dos últimos anos e para diferentes necessidades.

Outro aspecto, também de relevo mas menos explorado nesta área, diz respeito à utilização de novas tecnologias em contexto de sala de aula, de que se destacam por exemplo os *tablet PCs* e os *Personal Digital Assistants (PDAs)* (Anderson et al., 2004; Wilkerson et al., 2005; Denning et al., 2006). Os resultados parecem indicar que o uso de tecnologia educacional melhora a aprendizagem (Koile e Singer, 2006). Koile destaca o facto desta utilização permitir, entre outros, aumentar o centro de interesse dos alunos e a atenção na aula; fornecer *feedback* imediato quer ao aluno quer ao professor (sobre as dúvidas dos alunos); ajustar os materiais em tempo real de acordo com as dúvidas e respostas dos alunos ou aumentar a satisfação dos alunos.

Nesta área e de interesse particular para o tópico em estudo destaca-se também a utilização de robôs. Fagin e Merkle (2003) descrevem um estudo onde relatam a eficácia da utilização de robôs no ensino da programação. Refira-se também a importância do desenvolvimento do *Lego Mindstorm* gerador de variadas iniciativas (Lawhead et al., 2003) e diversos artigos (Harlan et al., 2001; Blank et al., 2003; Imberman e Klibaner, 2005).

1.2.6 Transferência de práticas profissionais para a sala de aula

A transferência da prática profissional para o contexto de sala de aula poderá ser outro caminho a seguir (Fincher e Petre, 2004). Os tópicos de *design patterns and frameworks*, *extreme programming*, *agile software development* e *test-driven development* são disso exemplo (Gamma et al., 1995; Fowler, 1999; Beck, 2000; Cockburn, 2002; Beck, 2003; Martin, 2003; Christensen, 2004). Refira-se também um *workshop* recorrente da conferência OOPSLA (*Object-Oriented Programming Systems, Languages and Applications)* que objectiva angariar participantes da indústria bem como do meio académico, de forma a promover discussões sobre a melhor maneira de ensinar *design patterns* em disciplinas introdutórias de programação (Alphonce, 2003). Outros têm questionado a pertinência de introduzir os *design patterns* nos cursos de iniciação à programação (Astrachan et al., 1998; Gelfand et al., 1998; Clancy e Linn, 1999; Nguyen e Wong, 1999; Preiss, 1999). Há também quem refira que a abordagem de iniciar um curso de programação com a estratégia de orientação a objectos deva ser modificada para a iniciação com *design patterns* (Pecinovský et al., 2006). No estudo conduzido por estes autores é também referido que os *frameworks* devem ser introduzidos o mais cedo possível de forma a treinar os alunos no sentido de reutilizarem código e não tanto de o produzirem. Refira-se também a importância da utilização do método *agile software development* para desenvolvimento de programas, constatado pela edição especial da *Computer Science Education* (Williams e Tomayko, 2002). Outro método considerado benéfico para uma melhor aprendizagem e desempenho de programação é o da utilização de *extreme programming* (Williams e Kessler, 2001; Bergin et al., 2004). Diversos educadores sugerem também a utilização de *test-driven development* para este efeito (Edwards, 2004; Jones, 2004; Janzen e Saiedian, 2006). Estes autores mencionam que em vez de os alunos utilizarem uma abordagem de tentativa e erro para encontrar os seus erros de programação, ao usarem esta metodologia desenvolvem um conjunto de competências, nomeadamente reflexão, compreensão, análise e teste de hipóteses.

1.2.7 Transferência do ensino presencial para o ensino à distância

Tal como muitas outras disciplinas, a informática está cada vez mais a ser ensinada à distância. Muitas das publicações sobre ensino/aprendizagem de programação relatam transformações dos cursos do modo presencial para sistemas de *e-learning*. Edwards refere um *workshop* em "*Establishing a Distance Education Program*" (Edwards et al., 2000) e Gersting relata experiências à distância realizadas em educação em ciências da computação, no Havai (Gersting, 2000). Bennedsen e Caspersen descrevem um curso introdutório de programação com suporte na *world wide web,* fundamentando as tecnologias utilizadas e o tipo de organização do curso a fim de possibilitar uma certa flexibilidade, para compensar os inconvenientes inerentes a este tipo de ensino (Bennedsen e Caspersen, 2003). Um dos pioneiros neste domínio foi o projecto *Runestone* (Daniels et al., 1999). No *Runestone*, os estudantes trabalhavam num projecto informático, em equipas e sob supervisão académica. Uma característica especialmente interessante deste projecto era o facto de metade dos estudantes em cada equipa ser da Suécia e metade dos E.U.A. Os estudantes de cada equipa viviam e trabalhavam em fusos horários diferentes e nunca se encontravam frente a frente. Contudo trabalhavam no mesmo projecto e eram avaliados como em qualquer outro trabalho académico similar. O objectivo primário do projecto era o de proporcionar uma experiência real internacional em CSE de uma forma valiosa para alunos e professores. Adicionalmente, os alunos experienciavam o trabalho em equipa com pessoas de diferentes contextos educacionais, culturais e linguísticos. O *Runestone* avaliava soluções técnicas e pedagógicas de colaboração investigando em particular a forma, quantidade e qualidade de aprendizagem num contexto de aprendizagem deste tipo.

1.2.8 Recrutamento e retenção

Esta é uma área que actualmente tem um grande foco. Uma das grandes questões nela tratada diz respeito à procura de indicadores de sucesso e justificações de insucesso em disciplinas de programação. Relativamente aos primeiros têm sido investigadas diversas variáveis, nomeadamente as habilitações matemáticas (Butcher e Muth, 1985; Bennedsen e Caspersen, 2005a; Ventura, 2005), o desempenho em disciplinas anteriores de programação (Chamillard, 2006), a capacidade de abstracção (Bennedsen e Caspersen, 2006), as percepções pessoais dos estudantes (Wilson e Shrock, 2001) ou factores emocionais (Cegielski e Hall, 2006). Mais especificamente têm sido desenvolvidos esforços no sentido de desenvolver instrumentos indicadores de aptidões para programar. Destaque-se, por exemplo, o CPAB (*Computer Programmer Aptitude Battery*) (Palormo e Fisher, 1964), o APTS (*Aptitude Profile Test*

Series) (Morgan et al., 2000) ou o teste desenvolvido por Dehnadi na Middlesex University (Dehnadi, 2006).

Relativamente às causas de insucesso, variados aspectos são estudados, incidindo principalmente nas taxas de desistência, na diversidade e em questões de género. Beaubouef e Mason (2005) investigaram as causas possíveis para as elevadas taxas de reprovação e desistência de estudantes de ciências da computação. Os autores invocam variados factores que culpam quer alunos quer professores, nomeadamente: o mau aconselhamento dos alunos no sentido de frequentarem cursos para os quais não têm aptidões; fracas competências matemáticas e de resolução de problemas dos alunos; fracas aptidões dos alunos relativamente a planeamento de projectos; disciplinas laboratoriais de programação mal planeadas; falta de tempo para fornecer *feedback* adequado aos alunos; professores mal preparados; má escolha da linguagem a ensinar, entre outros.

A questão da atractividade de disciplinas de programação para o sexo feminino é outra questão bastante em voga. Refira-se o *workshop* mencionado por Cuny e Aspray (2002) que reuniu um grupo de peritos para discutir o recrutamento e a permanência das mulheres em ciências da computação e em programas graduados de engenharia. Estudos atribuem o desinteresse do sexo feminino e desencorajamento desta temática à existência de estereótipos que consideram as disciplinas de programação como não criativas, associais, sem aplicabilidade, não amigáveis e intimidantes (Margolis e Fisher, 2003).

1.2.9 Construção da disciplina

Esta categoria diz respeito a questões sobre a constituição dos cursos de ciências da computação. Nesta temática são consideradas essencialmente questões sobre quais os conceitos basilares a ensinar/aprender, quais as disciplinas elementares, que princípios fundamentais devem ser abrangidos, que áreas curriculares são avançadas e quais são opcionais. Os currículos de computação da *Association for Computing Machinery* têm proporcionado discussões em torno destas questões. A *ACM's Education Board* foi estabelecida a fim de desenvolver recomendações sobre o currículo, estando a trabalhar neste assunto há aproximadamente 40 anos, resultando em várias recomendações de que se destacam: *Curriculum'68* (Atchison et al., 1968), *Curriculum'78* (Austing et al., 1979), *Computing Curricula 1991* (Tucker, 1991), *Computing Curriculum 2001* (ACM, 2001), *Computing Curriculum 2005* (ACM, 2005) e *Computing Curriculum 2008* (ACM, 2008). Actualmente, o Conselho de Educação da ACM formou quatro grupos de trabalho que abrangem a acreditação, recomendações curriculares, candidaturas e "tecnologias e ferramentas". Para além da construção do currículo, a área abrange temáticas que dizem respeito à natureza dos cursos de ciências da computação, conduzindo a discussões referentes ao tipo e quantidade

de assuntos/unidades curriculares que devem ser incluídos (Fincher e Petre, 2004). Um exemplo contemporâneo referente a este assunto é a coluna de Denning nas comunicações da ACM acerca da profissão de Tecnologias da Informação (Denning, 2001; Denning, 2002; Denning, 2003; Denning, 2004a; Denning e McGettrick, 2005). Como referências importantes relativas a esta temática podem ainda citar-se Comer et al. (1989), Dijkstra (1989), Simons et al. (1991), Wegner e Doyle (1996), Denning e Metcalfe (1997) e GCCER (2004). Orientações estratégicas acerca da educação em ciências da computação são também esboçadas em Tucker (1996). Mencione-se ainda a conferência mais recente sobre grandes desafios na educação de ciências da computação (McGettrick et al., 2005).

1.3 Objectivos do Estudo

As dificuldades existentes no ensino e aprendizagem das disciplinas introdutórias de programação são uma realidade. Face a este cenário, e sendo também professora na área, a autora deste trabalho encontra-se fortemente motivada a ajudar a descortinar as causas principais deste problema e procurar soluções que contribuam para o conseguir ultrapassar ou pelo menos minorar. Sendo assim, neste trabalho a preocupação principal consistirá em averiguar as principais dificuldades inerentes à aprendizagem inicial de programação, tentando compreender as razões subjacentes a este problema. Em função das causas detectadas, iremos propor uma estratégia de ensino/aprendizagem. Cremos que o problema é complexo, existindo uma multiplicidade de factores que contribuem para este cenário. O principal objectivo é o de capturar os factores mais importantes associados à prática do ensino e aprendizagem de programação, de forma a proporcionar práticas educativas adequadas conducentes à minimização do insucesso em disciplinas iniciais de programação. Cremos ser possível intervir no sentido de melhorar o panorama. Esta intervenção centrar-se-á necessariamente na análise do processo de ensino e aprendizagem de programação. Como tal é em torno do desempenho das personagens (professores e alunos) intervenientes neste processo e do próprio cenário (condições de ensino) que se farão os estudos. Estes terão como base algumas referências da literatura que consideramos pertinentes.

Pensamos que o problema está principalmente relacionado com três factores, nomeadamente, o conhecimento prévio dos alunos, os métodos de ensino e os métodos de aprendizagem. Estamos convictos de que um dos principais factores para as dificuldades mencionadas, factor impulsionador deste trabalho, está relacionado com os défices de competências que os alunos apresentam à entrada de uma disciplina de programação. Cremos que um dos grandes obstáculos impeditivos à resolução de um problema de programação está relacionado com a incapacidade de os alunos criarem algoritmos, causada

não apenas pela falta de conceitos básicos de programação, mas essencialmente pela incapacidade de resolver problemas em geral e em especial os que apresentam uma natureza matemática implícita ou explícita. Sendo assim, investigámos o seguinte factor:

F1 – Muitos alunos com dificuldades de aprendizagem de programação não possuem as competências necessárias exigidas por uma disciplina introdutória de programação.

Relativamente a este factor, consideramos que, entre outras, a muitos alunos faltam habilitações básicas fundamentais para obter sucesso a uma disciplina de programação, essencialmente porque:

F1.1 – Muitos alunos com dificuldades de aprendizagem de programação apresentam muitas dificuldades em resolver problemas.

F1.2 – Muitos alunos com dificuldades de aprendizagem de programação apresentam défices de conhecimentos matemáticos e lógicos.

Porém, pensamos que os alunos poderiam ser ajudados nas suas tarefas de aprendizagem, se as condições de ensino e aprendizagem fossem diferentes. Ainda que não possam ter as condições de ensino ideais, cremos que os professores podem proporcionar condições adequadas à forma preferencial de aprendizagem e ao desenvolvimento cognitivo do aluno. Consideramos este aspecto com influência nos resultados académicos, em especial em disciplinas exigentes que requerem cuidados peculiares no que concerne à forma de estudo e de ensino. Assim, relativamente aos métodos de ensino e ao seu principal actor (professor), consideramos o seguinte factor de investigação:

F2 – As condições e métodos de ensino habitualmente utilizados não são os mais adequados para o ensino de programação.

Consideramos que as condições em que o ensino decorre, com turmas demasiado grandes e a divisão das aulas em teóricas, teórico-práticas e/ou laboratoriais, não são adequadas ao ensino da programação. Este é um aspecto no qual não podemos intervir, mas antes fazer uma chamada de atenção. Porém, encaramos como de primordial importância a consideração dos estilos de aprendizagem como parte integrante dos métodos de ensino. Pensamos que a não consideração da variedade de estilos de aprendizagem em sala de aula, pode gerar falta de sintonia entre os estilos de aprendizagem dos alunos e os estilos de ensino do professor. Este factor poderá ser responsável pela falta de interesse de alguns alunos, o que em disciplinas particularmente difíceis como as programações, se poderá traduzir em desmotivação, baixo desempenho e em última instância em desistência. Estes resultados podem também, por consequência, originar frustração e desmotivação no próprio professor. De notar que o que se pretende verificar com este factor de investigação não é a

existência da disparidade de estilos de ensino e aprendizagem ou de metodologias desajustadas. Pretende-se sim caracterizar o público-alvo (alunos do 1º ano de um curso superior de Engenharia Informática/Ciências da Computação) em termos de estilos de aprendizagem, despertando, em consequência disso, os professores para esta realidade, eventualmente desconhecida da maioria deles. Face ao exposto, considerámos importante encetar estudos, no sentido de obter resposta às seguintes questões: Quais os perfis de estilos de aprendizagem predominantes nos alunos do 1ºano de uma licenciatura em Informática? Será que determinado perfil de aprendizagem está relacionado com os desempenhos a programação? Estas perguntas poderão ser traduzidas nos seguintes factores de investigação:

F2.1 – A Engenharia Informática/Ciências da Computação atrai alunos com determinado perfil de estilos de aprendizagem.

F2.2 – Existe correlação entre os estilos de aprendizagem dos alunos e os seus resultados à primeira disciplina de programação.

As nossas convicções e os estudos da literatura levam-nos também a considerar que o tipo de actividades normalmente propostas no contexto das disciplinas de programação não respeita os níveis de desenvolvimento cognitivo dos alunos. Cremos que o nível cognitivo dos alunos se encontra muito distanciado do nível de actividades propostas pelos professores, o que gera desmotivação conduzindo necessariamente ao fracasso. Face ao exposto e relativamente a F2 lançamos ainda o seguinte factor de investigação:

F2.3 – As tarefas habitualmente propostas pelos professores apresentam níveis de dificuldades desajustados ao nível cognitivo do aluno.

Também acreditamos que os alunos não utilizam métodos de estudo adequados às disciplinas de programação. Relativamente aos métodos de estudo e ao seu actor principal (aluno), consideramos o seguinte factor de investigação:

F3 – Os métodos de estudo utilizados pelos alunos não são os mais adequados para a aprendizagem em geral e da programação em particular.

Apesar de partirmos dos factores anteriormente apresentados, posteriormente, com o decorrer dos trabalhos acrescentámos ainda o seguinte factor de investigação que se afigurou relevante.

F4 – As percepções pessoais dos alunos são em geral muito baixas, insuficientes para enfrentar as exigências das disciplinas de programação.

A área de ensino e aprendizagem de programação inclui, actualmente, uma enorme diversidade de tópicos de pesquisa. Relativamente às dez áreas referidas neste capítulo demos particular atenção às questões relativas aos dois grandes actores do processo de ensino e aprendizagem, nomeadamente alunos e professores. Sendo assim, foi analisada literatura essencialmente relativa aos tópicos, Compreensão dos alunos e Métodos de ensino e Avaliação. Destas, a área mais pesquisada foi sem dúvida a referente à compreensão dos alunos, tendo sido feitas análises da literatura referentes a diversos tópicos, nomeadamente estudos psicológicos, concepções e *misconceptions* e comportamento dos principiantes em ciências da computação. Também concedemos alguma importância às questões da animação, visualização e simulação, métodos de ensino (nomeadamente, ensino centrado no aprendiz, de acordo com perspectivas construtivistas, não afastando outras abordagens para ensinar programação de que se destacam, o currículo invertido) e questões referentes à avaliação. Apesar de acharmos estes tópicos de elevada importância e interesse, a atenção principal teve de ser direccionada para os tópicos que poderiam proporcionar resultados o mais concreto e realísticos possível. Acima de tudo, tivemos de dimensionar o tempo para os aspectos que, na nossa óptica, eram mais pertinentes relativamente aos factores de investigação, nomeadamente com tónica na resolução de problemas, estilos de aprendizagem, métodos de estudo e motivação dos alunos.

Sendo assim, os factores de investigação motivaram a realização de um conjunto de estudos que permitiram averiguar certos aspectos a fim de edificar uma nova proposta de ensino e aprendizagem de programação incluindo não apenas um dos tópicos acima referidos, mas antes um conjunto daqueles que se revelaram pertinentes para uma nova abordagem de ensino/aprendizagem que possa ser bem sucedida.

Colocadas as questões e aspectos de investigação, o desafio deste trabalho foi, por um lado testá-las e, por outro, propor métodos de ensino e de estudo que possam melhorar o cenário apresentado. As teses referentes aos factores de investigação são difíceis, senão impossíveis de provar. No entanto, a extrema utilidade das reflexões proporcionadas levaram-nos a sugerir um conjunto de propostas. Essas propostas são concretizadas através de, sugestões de melhoria dos métodos de ensino, sugestões de melhoria dos métodos de estudo, sugestões para minorar as dificuldades dos alunos, promovendo o desenvolvimento das competências adequadas e propostas de estratégias para aumentar a motivação dos alunos.

As propostas têm em consideração uma base. Não é nossa convicção que a aplicação das propostas apresentadas ao longo deste trabalho permitam transformar qualquer aluno num programador brilhante. No entanto, pensamos que a prossecução das propostas apresentadas poderá fornecer uma boa estruturação de forma a que um aluno regular, desde

que fortemente motivado, possa compreender os conhecimentos básicos e aprender estratégias para realizar com sucesso as disciplinas introdutórias de programação.

Este documento encontra-se estruturado em quatro partes. A primeira, Capítulo 1 - Introdução, tem como objectivo, contextualizar a área em que se desenvolveu o trabalho documentado nesta tese e apresentar os principais objectivos e motivação deste trabalho. A segunda parte, inclui os capítulos teóricos de suporte ao diagnóstico das características do aluno e caracterização do trabalho do professor que, na nossa óptica, mais afectam o problema em estudo, bem como os capítulos teóricos de suporte às propostas de solução. Neste sentido, engloba os seguintes capítulos: Capítulo 2 - Resolução de problemas; Capítulo 3 - Estilos de aprendizagem; Capítulo 4 - Taxonomias e Capítulo 5 - Ferramentas de apoio ao ensino e aprendizagem de programação.

O Capítulo 2 – Resolução de problemas, justifica-se por pensarmos que uma das principais causas na origem das dificuldades de programação reside num défice do aluno relativo a capacidades de resolução de problemas. Desta forma, este capítulo tem como propósito abordar a temática "Resolução de problemas", referindo diferentes conceitos, teorias e modelos referentes às competências envolvidas na resolução de problemas. Este capítulo termina com um conjunto de recomendações genéricas a ter em consideração para melhorar a capacidade de resolução de problemas.

O Capítulo 3 – Estilos de aprendizagem, fundamenta a nossa convicção relativamente à importância dos métodos de ensino, nomeadamente, no que concerne às formas de expor, apresentar informações e propor actividades, como determinantes dos resultados da aprendizagem. Desta forma, este capítulo tem como propósito abordar a temática "Estilos de aprendizagem", referindo diferentes conceitos, teorias e modelos sobre este assunto. Este capítulo termina com um conjunto de sugestões genéricas que quer alunos quer professores devem ter em consideração para maximizar a aprendizagem.

O Capítulo 4 – Taxonomias, tem como desígnio apresentar diversas taxonomias para classificação de conhecimentos/competências, apresentando uma descrição mais detalhada da taxonomia considerada mais relevante para propor actividades de resolução de problemas de programação.

O Capítulo 5 – Ferramentas de apoio ao ensino e aprendizagem de programação, apresenta um conjunto de ferramentas computacionais, desenvolvidas para auxiliar o ensino e aprendizagem de programação. Nomeadamente, são destacadas, através de uma explicação abreviada, aquelas cujo interesse consideramos mais valioso para ajudar a solucionar o problema estudado.

A terceira parte, inclui os capítulos referentes aos estudos realizados e às propostas de solução. Neste sentido, engloba os seguintes capítulos: Capítulo 6 – Estudos realizados e Capítulo 7 – Proposta de ensino e aprendizagem de programação.

O Capítulo 6 – Estudos realizados, apresenta os estudos realizados e respectivo tratamento estatístico, com o objectivo de dar resposta aos factores de investigação formulados.

O Capítulo 7 – Proposta de ensino e aprendizagem de programação, apresenta um conjunto de propostas, baseadas nos estudos teóricos fundamentados na literatura e nos estudos práticos realizados.

Finalmente, a última parte, inclui o Capítulo 8 – Conclusões e Trabalho Futuro, onde são tecidas algumas conclusões gerais e considerações sobre o trabalho apresentado, bem como apresentadas sugestões para a sua continuação.

Cap. 2

Resolução de problemas

"Um problema não está necessariamente resolvido pelo facto da resposta correcta ter sido encontrada. Um problema não está verdadeiramente resolvido a menos que o aprendiz compreenda o que fez e conheça a apropriação das suas acções."

William A. Brownell

2.1 Introdução

Para aprender a programar não basta saber a sintaxe de determinada linguagem. Apesar de haver características específicas de determinada linguagem de programação que tenham de ser aprendidas, numa primeira fase a linguagem de programação deve ser apenas vista como um meio utilizado para resolver um problema de programação. Estamos convictos de que é a resolução de problemas, não apenas específicos de programação, mas também de carácter mais geral, que constitui uma das fontes criadoras de dificuldades numa fase inicial de aprendizagem de programação. Pensamos que as grandes dificuldades que os alunos sentem ao começar a programar estão relacionadas com a incapacidade de conceber algoritmos, e que esta se deve principalmente à incapacidade de resolver problemas. Admitimos que a resolução de problemas requer múltiplas competências, muitas vezes de diversos domínios, que deviam ter sido adquiridas ao longo dos anos de uma forma gradual. No entanto, pensamos que são as capacidades de resolução de problemas de ênfase matemática que se revestem de uma importância extrema para a capacidade de programar.

Este capítulo tem como propósito abordar a temática "Resolução de problemas", referindo diferentes conceitos, teorias e modelos existentes e nos quais baseamos o nosso estudo. No capítulo 5, são apresentados os estudos efectuados referentes à relevância da temática de resolução de problemas para a capacidade de programar.

2.2 Definições

De forma a melhor entendermos os processos envolvidos na resolução de problemas consultou-se a literatura, encontrando-se diferentes definições relativas ao conceito de problema. Para Gagné é um processo pelo qual o aprendiz descobre uma combinação de regras anteriormente aprendidas que ele pode aplicar para atingir uma solução para uma situação problemática nova (Gagné, 1965). Hayes (1981) definiu problema como a fenda que separa um estado presente de um estado desejado. Gil Pérez et al. (1988) consideram um problema como uma situação para a qual não há soluções evidentes. Perales (1993) considera-o uma situação qualquer que produz, de um lado, um certo grau de incerteza e, de outro, uma conduta em busca de uma solução. Para Mayer (1998) é um conceito multifacetado e complexo, com aspectos cognitivos, metacognitivos e motivacionais.

A maioria dos pesquisadores nesta área, encara a resolução de problemas como um estado subjectivo da mente, diferente de indivíduo para indivíduo, constituindo um desafio, uma situação não resolvida, cuja resposta não é imediata, que resulta em reflexão e uso de estratégias conceptuais e procedimentais, provocando uma mudança nas estruturas mentais. No entanto, concordamos com Flavell (1976) para quem a resolução de problemas se caracteriza pela complexidade e instabilidade, pela riqueza e criatividade, pelo que consideramos sensato não perder muito tempo a tentar fixar as suas propriedades numa definição formal.

2.3 Etapas

De forma a tentar perceber as dificuldades dos alunos relativas à resolução de problemas, fizemos uma pesquisa da literatura, baseando-nos particularmente no trabalho de Almeida (2004), sobre as fases envolvidas no processo de resolução de problemas em geral. Colhemos teorias de diferentes épocas e domínios do conhecimento.

Descartes (1693) apresentou quatro regras no seu discurso do método, nomeadamente:

i) Nunca aceitar algo como verdadeiro a menos que seja evidentemente reconhecido como tal, ou seja, evitar toda a precipitação e julgamento antecipado.

ii) Dividir cada uma das dificuldades em tantas partes quantas possíveis.

iii) Pensar de uma forma ordenada, começando com os aspectos mais simples e fáceis de compreender, direccionando-se gradualmente para os mais complexos.

iv) Fazer enumerações e revisões tão completas quanto possível, de forma a assegurar-se que nada é omitido.

Polya (1945) descreve quatro fases da resolução de problemas, nomeadamente:

i) Compreender o problema.

ii) Procurar uma situação semelhante.

iii) Planear a solução.

iv) Executar o plano.

v) Verificar a resposta.

A literatura também refere modelos de resolução de problemas, entre eles o identificado pela sigla IDEAL, proposto por Bransford e Stein (1984) que descreve os passos de resolução de problemas como um processo uniforme traduzido, nas etapas seguintes:

i) ***I****dentification*, identificação do problema.

ii) ***D****efinition*, definição do problema com precisão.

iii) ***E****xploration*, exploração de estratégias para alcançar a solução do problema (com base no conhecimento e experiências prévias).

iv) ***A****ction*, acção, no sentido de execução do plano delineado na fase anterior.

v) ***L****earn*, observação e aprendizagem do efeito das acções realizadas em função da avaliação dos resultados dessas acções.

Também no âmbito das teorias do processamento da informação têm sido propostos vários modelos correspondentes às etapas de resolução de problemas que, apesar de genericamente similares, envolvem um número variável de passos que, no seu conjunto, traduzem uma sequência pouco variável. De acordo com Sternberg e Davidson (1989) esses passos são:

i) Identificação do problema.

ii) Selecção da operação mental para solucioná-lo com êxito.

iii) Representação clara, interna e externa, da informação.

iv) Selecção de uma estratégia adequada.

v) Distribuição dos recursos disponíveis.

vi) Monitorização dos diferentes momentos da resolução do problema, ou consciência do que se fez, do que se está a fazer e do que se terá que fazer.

vii) Avaliação das soluções.

Os métodos de resolução de problemas são genericamente descritos numa sequência lógica de etapas (Osche, 1990) que retratam as fases relatadas pelos modelos de processamento de informação:

i) *Input*, em que o problema é percebido e é feita uma tentativa de compreensão da situação ou problema.

ii) Processamento, fase em que são geradas e avaliadas alternativas e seleccionada uma solução.

iii) *Output*, que inclui a planificação e a implementação da solução.

iv) Revisão, em que a solução é avaliada e são feitas modificações, se necessário.

Santucci (2000) sintetizou no acrónimo FARE as várias técnicas e modalidades de resolução de problemas que, fundamentalmente, retomam o modelo original de Polya, referindo as seguintes etapas:

i) **F**ocalizar pela criação, selecção e definição do problema, decidindo o que é necessário saber.

ii) **A**nalisar, pela recolha dos dados de referência, determinar os factores relevantes e gerar soluções alternativas (ou plano de acção).

iii) **R**esolver, por selecção da uma solução, desenvolvimento de um plano de actualização e empenho na organização para alcançar o resultado esperado.

iv) **E**xecutar, encontrando uma solução e controlando o seu impacto durante a implementação do plano (avaliação dos resultados).

Em termos gerais, o ciclo de resolução de problemas é composto pelas seguintes fases (Pretz et al., 2003):

i) Reconhecer ou identificar o problema.

ii) Definir e representar mentalmente o problema.

iii) Desenvolver uma estratégia de resolução.

iv) Organizar o conhecimento acerca do problema.

v) Consignar recursos mentais e físicos para resolver o problema.

vi) Monitorizar o progresso em direcção à finalidade.

vii) Avaliar a correcção da solução.

Também Almeida (2004) propõe um modelo que estabelece o processo de resolução de problemas constituído pelas seguintes fases:

i) Reconhecimento, definição ou identificação do problema.

ii) Análise do problema e geração de soluções alternativas.

iii) Desenvolvimento de planos, avaliação das alternativas e selecção de uma delas.

iv) Selecção e implementação efectiva da solução.

v) Avaliação e teste da solução.

Atendendo à revisão da literatura anteriormente referenciada e ao estudo publicado pelo relatório PISA 2003 (OECD, 2003), passamos de seguida a uma descrição sobre as etapas que devem ser seguidas na resolução de um problema, bem como os obstáculos que os alunos, normalmente, enfrentam em cada uma delas.

- Compreensão do problema. Esta é uma etapa de crucial importância para uma boa resolução de problemas, muitas vezes descurada ou a que se dedica tempo insuficiente. Acontece frequentemente que os alunos estão a "meio caminho" da tentativa de resolver o seu problema de programação, quando descobrem que não compreenderam realmente o problema em questão, e então têm que recomeçar mais uma vez. Na programação, onde os problemas são frequentemente "mal definidos", esta fase inclui precisamente definir o problema compreendendo todos os aspectos ambíguos ou não completamente explicitados. Inclui o modo como os estudantes compreendem um texto, um diagrama, uma fórmula ou uma tabela e retiram inferências dos mesmos; como relacionam informação de várias fontes; como demonstram compreensão de conceitos relevantes; como usam o conhecimento que já possuíam anteriormente para compreender a informação dada.

- Caracterização do problema. Inclui a maneira como os estudantes identificam as variáveis do problema e as suas inter-relações; como decidem quais as variáveis relevantes e quais as irrelevantes; como constroem hipóteses; como reconstituem, organizam, consideram e avaliam criticamente a informação contextual. Esta fase implica procurar problemas relacionados ou análogos, que já tenham sido

resolvidos ou que sejam do conhecimento do aluno. Este tipo de estratégias é poucas vezes utilizado pelos alunos ou por vezes mal utilizado. Os alunos recorrem pouco a conhecimento inter-relacionado, outras vezes utilizam como referência problemas aparentemente iguais mas que implicam estratégias de resolução muito diferentes.

- Representação do problema. Inclui o modo como os estudantes constroem representações tabulares, gráficas, simbólicas ou verbais; como aplicam uma representação externa previamente fornecida na resolução do problema ou como optam por formatos alternativos. Os alunos deveriam ser encorajados a representar os problemas, de forma verbal, gráfica ou outra que melhor descreva a sua forma de o interpretar e que melhor traduza o seu entendimento sobre a forma de o resolver.

- Resolução do problema. Para resolver um problema é necessário que as partes que se forem resolvendo anteriormente sejam agora ligadas num todo coerente e correcto. Independentemente das diferentes abordagens (*top-down*, *bottom-up*, …) utilizadas para resolver um problema há, nesta etapa, que haver um esforço e persistência da parte do aluno para relacioná-las e resolver a tarefa no seu todo. Acontece frequentemente que os alunos desistem à mínima dificuldade, pondo de parte um problema pelo facto de não saberem solucionar uma das suas partes.

- Reflexão sobre a solução. Muitas vezes o grande desejo do aluno reside apenas na forma mais rápida de conseguir chegar a uma solução, não havendo posteriormente o cuidado de a analisar. Esta etapa deve incluir a análise cuidada da solução construída implicando a procura de informação adicional ou uma clarificação da mesma. Deve também ser considerada a avaliação da solução de diferentes perspectivas, numa tentativa de reestruturar a solução e de a tornar mais aceitável e optimizada. Consideramos que se aprende a programar não apenas programando, mas estudando e reflectindo sobre a forma como se programou. Será que o faríamos de forma diferente na vez seguinte? Como é que outros resolveram o mesmo problema? Em disciplinas de programação, as várias soluções construídas pelos estudantes deviam ser discutidas na turma, de modo a que todos pudessem verificar diferentes pontos de vista, partilhando estratégias e dificuldades.

- Comunicação da solução do problema. Inclui o modo como os estudantes seleccionam os meios e as representações adequadas à expressão e à comunicação das suas soluções a uma audiência externa. Pensamos que o acto de tentar

comunicar uma solução pode ajudar a melhor detectar problemas anteriormente não percepcionados.

De forma a averiguar os aspectos ou fases mais problemáticas no processo de resolução de problemas de programação, pesquisámos também a literatura, sem que nada de muito concreto fosse encontrado. Porém, Almeida (2004) apresenta um estudo referente à resolução de problemas. Apesar do público-alvo deste estudo consistir em alunos do ensino secundário e o domínio do conhecimento ser referente a Matemática, fornece um conjunto de indicações que parecem confirmar as nossas conjecturas. Assim, esta investigadora conclui que em apenas 50% dos casos os sujeitos definiram o problema, ou seja, identificaram as variáveis e a incógnita ou os estados inicial e final do problema. Em 32% dos casos identificaram apenas parte dos dados, das condições ou da incógnita. Aquando da planificação da resolução, em apenas 47% dos casos o problema foi devidamente equacionado. Nesta etapa da resolução sobressaiu a dificuldade de representação gráfica do problema numa qualquer forma abreviada e esclarecedora, sendo que essa representação foi feita, na maior parte das vezes, por uma qualquer notação ou esquema e, apenas na sequência de sugestão por parte da observadora das experiências.

No mesmo estudo, foi observado que na etapa da resolução propriamente dita, muitas vezes ocorrida logo após o contacto com o enunciado do problema, sobressaiu a pressão sentida pelos sujeitos, na sua globalidade, no sentido de encontrarem de imediato uma resposta. Também nós, na prática lectiva, verificamos que os alunos se precipitam frequentemente na tentativa de obtenção da solução antes de um completo entendimento do enunciado. No estudo referido era frequentemente recomendado pela observadora ponderação na obtenção das respostas, mas ainda assim surgiram 9% das respostas por *insight*, nem sempre correctas. Também se verificou que eram poucos os casos em que se percebia uma atitude reflexiva dos sujeitos. Adicionalmente, foi notório o fraco autocontrolo ou regulação dos componentes, em termos estratégicos, de planificação e verificação. Quanto ao aspecto de avaliação da resolução ou verificação do processo, registaram-se 45% de casos em que os sujeitos aceitaram a solução, bastando para a sua justificação, o facto de terem obtido um resultado, sem a preocupação da sua correcção ou coerência. Os investigadores deste estudo concluíram também que os sujeitos que melhor resolveram problemas foram os que melhor compreenderam os enunciados, tendo sido estes os que mais frequentemente verificavam, de forma controlada, o processo e progresso de resolução, ou a coerência da solução com os dados e condições do problema.

2.4 Competências

Para além das etapas recomendadas para uma correcta resolução de problemas, um aspecto importante para a capacidade de resolver problemas consiste em averiguar quais as competências subjacentes a esse processo, salientando que vários autores referem várias competências como requisitos. A hipótese de organização taxonómica das competências cognitivas pressupõe a consideração da mente como um universo hierárquico e multidimensional, cujos diferentes níveis de organização obedecem a regras diferentes (Demetriou, 1998). Neste sentido, a resolução de problemas é considerada a competência mais elevada de um contínuo de complexidade (Gagné, 1985; Seamster et al., 1997).

Também de acordo com o relatório PISA 2003, o acto de resolver problemas é uma amálgama de vários processos cognitivos diferentes, orquestrados no sentido de atingirem um certo objectivo que não poderia ser atingido, pelo menos de modo evidente, simplesmente aplicando um procedimento, um processo, uma rotina ou um algoritmo, já conhecido, de uma única área disciplinar. A resolução de problemas envolve diferentes competências de raciocínio. Assim o processo de resolução de problemas remete não só para as bases de conhecimento do sujeito que está a resolver o problema como também para as suas competências de raciocínio. Por exemplo, ao tentar compreender uma situação problemática, o indivíduo pode ter de fazer a distinção entre factos e opinião. Ao formular uma solução, pode ter de identificar relações entre variáveis. Ao seleccionar uma estratégia, pode ter de considerar a causa e o efeito. Ao comunicar os resultados, pode ter de organizar a informação de um modo lógico. Todas estas actividades requerem frequentemente competências de raciocínio analítico, de raciocínio quantitativo, de raciocínio analógico e de raciocínio combinatório. Estas competências de raciocínio formam o núcleo das competências de resolução de problemas.

Segundo o relatório PISA 2003 os diferentes raciocínios implicam diferentes actividades em diferentes situações. Assim, o raciocínio analítico é caracterizado por situações em que o indivíduo em aprendizagem tem de aplicar princípios da lógica formal, quando determina as condições necessárias e as suficientes ou quando determina se a implicação de causalidade ocorre no âmbito dos constrangimentos e das condições fornecidas no enunciado do problema. O raciocínio quantitativo é caracterizado por situações em que o indivíduo em aprendizagem tem de aplicar propriedades e procedimentos relacionados com a percepção do número e com as operações numéricas, para resolver determinado problema. O raciocínio analógico é caracterizado por situações em que o indivíduo em aprendizagem tem de resolver um problema inserido num contexto semelhante ao de um problema que lhe é familiar ou que inclui uma base problemática que o

mesmo tenha resolvido no passado. Nesta situação, os parâmetros ou o contexto do novo enunciado foram modificados, mas os factores de condução ou o mecanismo causal são os mesmos. O indivíduo deve ser capaz de resolver o novo problema, interpretando-o à luz da experiência passada, relativamente à situação análoga. O raciocínio combinatório é caracterizado por situações em que o indivíduo em aprendizagem tem de examinar vários factores, considerar todas as combinações em que estes podem ocorrer, avaliar cada uma destas combinações individuais, em relação aos requisitos, e depois seleccionar ou ordenar hierarquicamente as combinações.

Kizlik (2010) identificou um conjunto de competências que considerou nucleares para um desempenho cognitivo eficaz, nomeadamente: competências de focagem, competências de recolha da informação, competências de memória, competências de organização, competências de análise, competências de execução, competências de integração e competências de avaliação. Kizlik argumenta ainda que, quanto mais os indivíduos forem capazes de integrar as diferentes competências, mais eficaz será o seu comportamento de resolução de problemas. As competências menos desenvolvidas, por sua vez, indicarão aspectos a ter em consideração num qualquer programa de treino ou aquisição de resolução de problemas.

No entanto, concordamos com Sloane e Linn (1988) e Lister et al. (2004) quando afirmam que programar não consiste numa única competência, nem num conjunto de competências, mas antes que as competências formam uma hierarquia e o programador tem de usar muitas delas em simultâneo. Que competências são essas? Que competências possuem os peritos ao resolverem problemas que os inexperientes não têm? De forma a promover a melhoria da resolução de problemas pelos alunos, pensámos ser importante compreender as vantagens que os especialistas apresentam relativamente aos inexperientes, de forma a poder transformar essas vantagens em directrizes para que os novatos aprendam a melhor forma de resolver problemas.

Costa e Moreira (1996) fazem uma análise comparativa entre a forma de resolução de problemas por principiantes e especialistas em diferentes domínios do conhecimento, nomeadamente, Química, Física, Matemática e Genética. De um modo geral, os comportamentos dos principiantes e especialistas relativos à resolução de problemas foram diferenciados quanto à maneira como os conceitos e princípios são armazenados e recuperados na memória de longo prazo, bem como quanto ao conjunto de estratégias utilizáveis pelos sujeitos em função do seu conhecimento e experiência (prática). As conclusões gerais a que chegaram estes e outros autores, relativas às diferenças de comportamento entre especialistas e inexperientes, são de seguida apresentadas:

- Os especialistas trabalham "para a frente", constroem uma representação mais completa do problema pelo seu conhecimento extra disponível e os inexperientes trabalham "para trás" na tentativa de encontrar as incógnitas do problema (Larkin et al., 1980; Heller e Reif, 1984; Dufresne, 1988).

- Os inexperientes tendem a agrupar problemas que tenham as mesmas características superficiais, por semelhança de enunciados ou forma, através de tarefas de lembrança enquanto que os especialistas agrupam problemas por princípios, através de tarefas de raciocínio (Larkin et al., 1980; Chi et al., 1981; Smith e Good, 1984; Hardiman et al., 1989).

- Os novatos tendem a considerar a primeira opção que surge como válida, não testar hipóteses, usar mal a lógica, ignorar raciocínios anteriores, não reflectir sobre o resultado (Smith, 1988).

- Os inexperientes tentam imediatamente resolver o problema, resolvem-no muitas vezes por acaso numa sequência linear e geralmente não possuem muito conhecimento auxiliar. Os especialistas dão uma atenção especial à análise baseada no seu conhecimento tácito e experiência, resolvem problemas por métodos de refinamento, possuem conhecimento auxiliar numa forma mais implícita (Larkin e Reif, 1979).

- Os especialistas aproximam-se da solução através de um processo de refinamentos sucessivos, começam com uma descrição grosseira dos problemas, em palavras e desenhos e só então examinam os detalhes do problema e introduzem a Matemática necessária. Os inexperientes começam por uma procura do princípio ou equação que resolva o problema, não tendo o conhecimento inter-relacionado como os especialistas (Tobón e Perea, 1985).

- Os bons solucionadores de problemas, resolvem problemas partindo de um princípio geral, usando o raciocínio dedutivo (Rosa et al., 1992).

- Os especialistas tendem a perceber um problema como uma tarefa de análise e raciocínio enquanto que os inexperientes tentam encontrar uma resposta rápida para o problema (Smith e Good, 1984).

- Os inexperientes enfatizam o nível quantitativo enquanto que os especialistas trabalham primeiro em termos qualitativos (Slack e Stewart, 1990).

Apesar de as pesquisas realizadas sobre esta temática se reportarem essencialmente a estudos nas áreas da Física e da Matemática, o nosso objectivo era o de encontrar trabalhos

similares na área da programação. Será que as competências necessárias para resolver problemas de programação são as mesmas necessárias à resolução de problemas de outras áreas do conhecimento, como as anteriormente mencionadas? Pesquisámos a literatura e sobre o assunto encontrámos algumas referências quanto à forma diferenciada pela qual os inexperientes e os especialistas resolvem problemas. Considerámos este aspecto de particular importância a fim de poder dar sugestões para que os inexperientes possam evoluir no sentido de se tornar especialistas. Pudemos extrair as conclusões de seguida apresentadas.

Os inexperientes:

- podem compreender a sintaxe e semântica de declarações individuais mas não conseguem combiná-las em programas para resolver determinado problema, faltando-lhe um modelo mental adequado da área (Kessler e Anderson, 1989).
- estão limitados a um conhecimento superficial do assunto, apresentando um conhecimento frágil (algo que os alunos conhecem mas que falham quando necessitam de o aplicar) e negligenciam estratégias (Perkins e Martin, 1986).
- usam estratégias inadequadas de resolução de problemas, ou seja, copiam uma solução aparentemente similar em vez de usarem estratégias dependentes do problema particular (Perkins e Martin, 1986).
- tendem a abordar a programação através de estruturas de controlo, usando uma aproximação linha-a-linha para a solução dos problemas (Anderson, 1985).

Os especialistas:

- possuem vários modelos mentais escolhendo-os e misturando-os de forma oportunística, aplicando todo o conhecimento que possuem (Visser e Hoc, 1990).
- têm um conhecimento profundo do assunto, o qual é hierárquico e em camadas com mapeamentos explícitos entre elas. (Visser e Hoc, 1990).
- quando lhes é fornecida uma tarefa numa área que lhes é familiar, trabalham "para a frente" a partir dos elementos fornecidos, desenvolvendo subobjectivos de uma forma hierárquica. Porém, quando lhes fornecem um problema desconhecido, recorrem a técnicas gerais de resolução de problemas (Chi et al., 1981; Davies, 1990).
- possuem uma forma mais aprimorada de reconhecer problemas que requerem uma solução similar (Chi et al., 1981; Davies, 1990).

- abordam a programação através das suas estruturas de dados ou objectos (Petre e Winder, 1988).
- usam algoritmos em vez de uma sintaxe específica (abstraem-se da linguagem particular para o conceito geral) (Petre e Winder, 1988).
- são mais rápidos e cuidadosos na obtenção de soluções para os problemas (Allwood, 1986; Wiedenbeck, 1986).
- possuem um melhor conhecimento sintáctico e semântico e melhores competências tácitas e estratégicas (Bateson et al., 1987).
- veêm os blocos de código como instâncias de problemas bem conhecidos (Bonar, 1982; Soloway et al., 1982; Johnson et al., 1983; Ehrlich e Soloway, 1984).

Dreyfus et al. (1986) descrevem uma sequência de cinco estágios de aquisição de competências, ao longo de um contínuo (de inexperientes até especialistas). A sua abordagem é completamente geral e não destaca qualquer tipo particular de competências. Sendo assim, apresenta a seguinte caracterização:

- Inexperiente: aprende factos objectivos, características e regras para determinadas acções baseadas em factos e características (tudo o que faz é independente do contexto).
- Iniciante avançado: começa a reconhecer e manipular situações não cobertas por factos fornecidos, características e regras (sensíveis ao contexto) sem muita compreensão sobre o que faz.
- Competente: após considerar a situação como um todo, escolhe conscientemente um plano organizado para alcançar o objectivo.
- Proficiente: não tem que passar conscientemente por todos os passos para determinar um plano para alcançar o objectivo.
- Especialista: geralmente sabe o que fazer com base em compreensões amadurecidas e treinadas.

Também no livro "*How people learn*" (Bransford et al., 2000), os autores consideram os seguintes princípios-chave relativamente ao conhecimento dos especialistas e às suas potenciais implicações para a aprendizagem:

- Os especialistas encontram características e padrões significativos que não são notados pelos inexperientes.

- Os especialistas adquirem uma notável quantidade de conhecimento, organizado de forma a reflectir uma profunda compreensão do assunto.
- O conhecimento dos especialistas não pode ser reduzido a conjuntos de factos isolados ou proposições, mas antes reflecte contextos de aplicabilidade, ou seja, o conhecimento é condicionado por um conjunto de circunstâncias.
- Os especialistas são capazes de, com grande flexibilidade, extrair aspectos importantes do seu conhecimento através de pouco esforço de atenção.
- O facto de os especialistas conhecerem os seus assuntos minuciosamente não garante que sejam capazes de o ensinar aos outros.
- Os especialistas têm vários níveis de flexibilidade ao abordarem novas situações.

Pensamos que a transformação de um inexperiente num perito de resolução de problemas passa também pela compreensão mais profunda das dificuldades que os alunos inexperientes com défices nesta área apresentam. Porém, este tipo de estudo bem como a investigação de técnicas e estratégias para transformação de um inexperiente num perito de resolução de problemas sai fora do âmbito deste trabalho. Contudo, como consideramos o aspecto de resolução de problemas de extrema importância, constituindo mais um factor para a possível obtenção de sucesso em qualquer disciplina que necessite deste tipo de competências, finalizamos este capítulo com uma secção de recomendações no sentido de futuras explorações.

2.5 Reflexões/Sugestões

Consideramos que o conjunto de etapas relativas ao processo de resolução de problemas referidas na literatura, se praticadas ao programar, conduzirão à sistematização e disciplina, que a maioria dos alunos não tem, podendo dar ao aluno inexperiente algumas ideias sobre como estruturar o seu labirinto de pensamentos sobre algo que se começa a assemelhar a um algoritmo.

Porém, a aplicação da sequência de passos descrita pode ser coadjuvada se os resolvedores dominarem e utilizarem um conjunto de técnicas facilitadoras. A literatura refere várias estratégias, entre elas a tentativa e erro; a redução das diferenças; a análise meios-fins; a analogia; a imagem mental; a simulação, técnicas focalizadas em pensamento lógico e crítico; técnicas centradas em pensamento criativo, lateral ou divergente. Adicionalmente e segundo Huitt (1992) é igualmente necessário identificar técnicas

específicas que acomodem preferências individuais. Cada indivíduo tem uma forma preferencial de receber e processar informação. Assim, determinadas técnicas podem beneficiar determinado perfil de aprendizagem enquanto outras podem servir melhor outros indivíduos caracterizados com perfis diferentes. Assim, a sugestão de determinada técnica para aplicação na resolução de determinado problema deverá também ter em atenção o perfil de estilos de aprendizagem do aluno em causa.

As nossas crenças baseiam-se na convicção de que um dos principais componentes na aprendizagem de programação consiste na organização das habilidades de resolução de problemas, elemento comum à construção de conhecimento em qualquer outro domínio. Como tal, um investimento nesta área que consideramos ser pertinente passaria por averiguar como tais habilidades são construídas, a fim de detectar e desenvolver aquelas em falta.

No que respeita à programação, apesar de encontrámos poucas referências sobre o assunto na literatura, consideramos interessante a utilização do Método Clínico Piagetiano (Delval, 2002) aplicado à resolução de problemas de programação. Este método consiste na realização de entrevistas individuais com os estudantes durante a resolução de um problema, elaboradas segundo os critérios de resolução que se desejam estudar. As diversas situações utilizadas e propostas por Piaget neste seu método envolviam manipulações concretas, embora também implicassem muitos processos de raciocínio lógico, indutivo e de representação. Foi realizado um estudo-piloto (Neto et al., 2006), direccionado à programação, onde se fizeram algumas adequações do método clínico piagetiano original. Estas modificações, foram consideradas fundamentais, pelo facto do objecto manipulado (programas) ser algo abstracto e o método prever situações mais concretas. O estudo piloto consistiu em observações externas durante as etapas de codificação dos programas, observações internas (anotadas pelos próprios estudantes) antes e depois de resolução dos problemas e entrevistas individuais. Apesar deste tipo de estudo permitir registar as dificuldades concretas dos alunos, a realização de um estudo deste tipo implicaria a existência de uma equipa alargada para a realização de uma investigação profunda, durante um período de tempo alargado.

À falta de condições ou instrumentos mais adequados pensamos ser pertinente a intervenção do professor, no sentido de promover nos seus alunos a capacidade de resolução de problemas de programação. Neste sentido, julgamos que o professor deverá, em qualquer actividade de programação, orientar os alunos para que sigam os passos recomendados na literatura para uma correcta resolução de problemas. Consideramos também de extrema importância que o professor explicite claramente determinado conjunto de procedimentos, referentes à estratégia utilizada em cada resolução. Sempre que possível, deverá também

tentar perceber a razão de determinada estratégia escolhida pelos seus alunos, avaliando as suas incorrecções e incompreensões.

2.6 Conclusões

A experiência pessoal, reforçada pelos relatos existentes na literatura, deixa-nos convictos de que a capacidade de resolução de problemas é um aspecto crucial para conseguir programar. Apesar de procurarmos definições sobre o conceito e as fases necessárias à correcta resolução de problemas pensamos que as conceptualizações teóricas, só por si, não serão suficientes para uma eficiente aplicação prática que potencie a sua aprendizagem. Pensamos ser necessária a existência de instrumentos e técnicas para adequadamente compreender um sujeito a resolver problemas genéricos e posteriormente, problemas específicos de programação. Esta compreensão permitiria diagnosticar as fases de resolução de problemas em défice ou menos conseguidas. Outra possibilidade poderia consistir no estudo e observação das diferenças entre grupos de indivíduos bons resolvedores de problemas *versus* fracos resolvedores. Independentemente do instrumento ou técnica de diagnóstico, o importante é que os alunos possam ser analisados sobre este tipo de capacidades para que posteriormente possam ser intervencionados através de programas de treino eficazes.

O objectivo deste trabalho não é o de estudar o tipo de défices de resolução de problemas dos alunos, mas antes testá-lo como factor causador dos maus resultados a programação. Porém, consideramos um interessante elemento de investigação a definição de um conjunto de funções cognitivas ou competências que serviriam de base a uma boa capacidade de resolução de problemas de programação. Tendo definidas essas competências, seria também importante averiguar as suas dependências a fim de verificar se seriam necessárias todas ao mesmo tempo ou em etapas distintas ou quais é que se deviam treinar primeiro. O estabelecimento de uma hierarquia de competências permitiria definir programas para diagnosticar e posteriormente treinar essas funções.

Na área da Psicologia facilmente se encontram definidos modelos/teorias cognitivas referentes ou implicados na resolução de problemas. Duas das áreas que têm fornecido valiosos contributos para o entendimento dos processos cognitivos são a Psicologia Cognitiva (processamento da informação) e a Psicologia do Desenvolvimento (desenvolvimento da inteligência). Sugere-se então a pesquisa e análise de instrumentos de avaliação cognitiva credíveis, de forma a detectar e posteriormente treinar as competências pretendidas.

Cap. 3

Estilos de aprendizagem

"O que eu ouço, esqueço. O que eu vejo, lembro. O que eu faço, aprendo."

Confúcio

3.1 Introdução

Os alunos aprendem de diversas formas - vendo e ouvindo, reflectindo e agindo, raciocinando lógica e intuitivamente, memorizando e visualizando, arranjando analogias e construindo modelos matemáticos, tendo consequentemente preferências distintas no que concerne à recepção da informação e consequente processamento em conhecimento. Os métodos dos professores também variam. Uns usam métodos mais expositivos, outros utilizam demonstrações e promovem discussões, uns concentram-se em princípios enquanto outros em aplicações, uns enfatizam a memória e outros a compreensão. Porém, pensamos que na maioria das salas de aula, mesmo com o processo de Bolonha em vigor, os alunos ainda são "forçados" a uma aprendizagem uniforme, devendo aprender ao mesmo ritmo e de acordo com as estratégias pedagógicas do professor. No entanto, é importante reconhecer que os alunos são diferentes, tendo cada um o seu próprio modo de receber e processar informações, resolver problemas e expor ideias, significando que cada um tem a sua própria personalidade e estilo de aprendizagem característico.

Pensamos que há um conjunto de factores que podem afectar a aprendizagem, tendo os métodos de ensino, as formas de expor e apresentar informações, eventualmente, impacto nos resultados da aprendizagem de cada aluno. Dada a diversidade de alunos, pensamos ser crucial a reflexão referente aos estilos de aprendizagem para atender às diferentes preferências de aprendizagem. No entanto, para tal é necessário que os professores tenham um melhor entendimento da aprendizagem em si, compreendendo a diversidade de meios pelos quais os

alunos aprendem, proporcionando um melhor relacionamento e entendimento com os seus alunos, no sentido de melhor planear o processo de ensino, utilizando estratégias educativas mais adequadas que se traduzam numa aprendizagem mais eficaz.

3.2 Conceitos

Desde os anos 70 que a comunidade educativa tem demonstrado interesse considerável relativamente à temática dos estilos de aprendizagem. Ao longo dos anos têm sido propostos numerosos modelos e realizadas pesquisas em diferentes contextos educativos. No entanto, o conceito de estilos de aprendizagem permanece um tópico controverso. Apesar da generalidade dos autores concordar que os indivíduos aprendem de diferentes formas, existem diferentes opiniões acerca da relevância, validade e aplicação das teorias subjacentes aos diferentes modelos de estilos de aprendizagem.

De acordo com Baleche (2003), os estilos de aprendizagem não se referem à forma como um indivíduo aprende, mas antes à forma como ele se comporta durante a aprendizagem. Assim sendo, como cada indivíduo é único, é possível a coexistência em sala de aula de múltiplos comportamentos no que concerne à forma de aprender. É então fundamental o conhecimento deste aspecto, por parte do professor, de forma a melhor responder a esta diversidade.

Há autores que fazem a distinção entre estilos de aprendizagem e estilos cognitivos, outros porém consideram que os dois termos se fundem tendo a mesma significação. De acordo com Riding e Cheema (1991), o termo estilo cognitivo foi definido por Allport em 1937 como um modo típico de determinado indivíduo resolver problemas, pensar, perceber e recordar.

De acordo com Keefe (1982), estilos de aprendizagem são características cognitivas, afectivas e comportamentais, que servem como um indicador relativamente estável de como o aprendiz percebe, interage e responde ao ambiente de aprendizagem. De Bello (1990) defende que estilos de aprendizagem são a maneira pela qual um indivíduo absorve, processa e retém uma informação nova e difícil. Para Dunn et al. (1990) estilos de aprendizagem são as formas pelas quais o aprendiz se começa a concentrar, processa e retém novas e difíceis informações. Os mesmos autores afirmam também que essa interacção ocorre de forma diferenciada para cada aprendiz e que os estilos podem mudar em qualquer momento como resultado da maturação.

Segundo Pennings e Span (1991) tanto os estilos cognitivos quanto os estilos de aprendizagem se referem à forma e não ao conteúdo do perceber, pensar, lembrar, aprender ou decidir. Porém, enquanto que os primeiros se relacionam com as estratégias utilizadas por cada indivíduo no processamento da informação para a resolução de problemas, os estilos de aprendizagem referem-se ao modo como o indivíduo interage com as condições de aprendizagem, incluindo os aspectos cognitivos, afectivos, físicos e ambientais que favorecem a aprendizagem. Outros autores consideram que tanto os estilos cognitivos como os estilos de aprendizagem se referem à forma como a mente processa a informação ou é influenciada pelas percepções de cada indivíduo (Messick, 1969; Coop e Brown, 1970).

É importante mencionar que qualquer dos estilos não implica habilidade, capacidade ou inteligência. Isto significa que não há estilos bons ou maus, melhores ou piores, há apenas diferentes estilos (Cerqueira, 2000).

3.3 Modelos

Na literatura encontram-se descrições de vários modelos de estilos de aprendizagem que foram desenvolvidos para classificar as preferências do aprendiz relativamente ao processo de entendimento e processamento da informação.

Um modelo de estilos de aprendizagem classifica os indivíduos de acordo com a forma como se encaixam num determinado número de escalas referentes aos diversos modos de receber e processar informação. Porém, existem numerosos modelos para determinar os estilos de aprendizagem de um indivíduo. Esta é uma área que está longe de consenso. Avaliar diferentes modelos de estilos de aprendizagem e as suas implicações para a pedagogia, requer uma apreciação complexa e muitas vezes controversa. De acordo com as áreas (psicologia, sociologia, educação, estudos políticos, entre outras) de proveniência dos investigadores as evidências são interpretadas de diferentes formas, tal como as teorias, utilizando diferentes termos.

Um importante trabalho a este respeito foi conduzido por Coffield et al. (2004), tendo os autores revisto 71 modelos de estilos de aprendizagem, considerando 13 como sendo os principais. O objectivo desse estudo foi o de avaliar os principais modelos de estilos de aprendizagem, discutindo as suas implicações e relevância para o ensino e aprendizagem. Esses autores referem que o facto de apenas terem considerado 13 dos 71 modelos para uma análise profunda e detalhada, não significa necessariamente que os modelos excluídos sejam fracos, mas que foram rejeitados devido a determinado critério estabelecido nesse estudo. Os critérios usados para seleccionar determinado modelo, estavam essencialmente relacionados

com as suas teorias de suporte, o facto do modelo se revelar original e crucial para a área no seu todo, conduzindo a futuras investigações por parte de outros investigadores e ser representativo da totalidade dos modelos disponíveis.

Os autores deste estudo classificaram os diferentes modelos em 5 famílias, de acordo com as crenças base acerca da aprendizagem, conceitos-chave e definições que ligam os principais pensadores influentes desse grupo. Os modelos, no seu todo, são baseados numa grande variedade de disciplinas, embora a dominante seja a psicologia cognitiva. Figuras influentes como Jean Piaget, Carl Jung e John Dewey deixaram marcas no trabalho de diferentes grupos de teóricos dos estilos de aprendizagem. O enquadramento em famílias baseia-se em diferentes pressupostos genéricos, por exemplo, na crença de que os estilos de aprendizagem e preferências são de um modo geral impostos biologicamente, significando que os estilos são relativamente fixos e que os métodos de ensino devem ser alterados para os acomodar. Outros defendem que os estilos de aprendizagem assentam profundamente em características da estrutura cognitiva, incluindo "padrões de capacidades", não sendo particularmente susceptíveis de ser treinados. Outros baseiam-se na crença de que os estilos de aprendizagem são um componente de um tipo de personalidade relativamente estável, relacionando-os com os traços de personalidade que modelam todos os aspectos de uma interacção individual com o mundo. Outros ainda defendem que um estilo de aprendizagem não é um traço de personalidade fixo, mas uma preferência diferencial para a aprendizagem, a qual muda ligeiramente de situação para situação, mas com alguma estabilidade de longo prazo.

Na secção seguinte descrevemos com algum detalhe o modelo de Myers-Briggs que, embora sendo um modelo para identificar os diferentes tipos de personalidade, constituiu um marco importante de suporte para a elaboração de variados modelos de estilos de aprendizagem. Com base no pressuposto de que o conhecimento dos tipos de personalidade ajudaria a um melhor entendimento das diferenças individuais, possibilitando a aplicação construtiva de estratégias de aprendizagem consonantes, foram desenvolvidos vários modelos referentes aos estilos de aprendizagem.

Descrevemos também dois proeminentes modelos de estilos de aprendizagem que, tanto quanto nos pudemos aperceber, foram os mais amplamente usados com estudantes universitários e em especial com alunos dos cursos de engenharia. Esses modelos referem-se à teoria da aprendizagem experiencial de David Kolb e ao modelo de estilos de aprendizagem de Felder-Silverman.

3.3.1 Modelo de Myers-Briggs

O modelo de Myers-Briggs (Myers e Myers, 1980), apesar de evidenciar os tipos de personalidade, também tem sido usado para medir os estilos de aprendizagem, uma vez que muitas categorias por ele definidas são baseadas em conceitos de natureza cognitiva. Desenvolvido por Isabel Briggs Myers e Katharine Cook Briggs, classifica os tipos de personalidade baseando-se nas teorias de Carl Jung.

Carl Jung dedicou grande parte da sua vida a analisar a personalidade humana, desenvolvendo uma das mais abrangentes teorias para explicar o desenvolvimento da personalidade do ser humano. Um importante contributo do seu trabalho encontra-se expresso no livro "*Psychological Types*" (Jung, 1971). Neste, Jung trata a problemática das diferenças individuais e dos tipos, salientando a compreensão que cada indivíduo deve ter de si mesmo e dos outros. Jung define os tipos psicológicos da seguinte forma:

"Tipo é um modelo característico de uma atitude geral que se manifesta em muitas formas individuais. Das muitas e possíveis atitudes, saliento […] quatro, isto é, aquelas que se orientam sobretudo pelas quatro funções psicológicas básicas: pensamento, sentimento, intuição e sensação. Quando uma dessas atitudes é habitual e imprime ao carácter do indivíduo um cunho determinado, falo então de tipo psicológico".

Em traços muito gerais, Jung considera como elemento de maior importância a inclinação natural do indivíduo, a sua preferência individual por um ou outro processo mental, por uma ou outra maneira de funcionar como ser humano. Mediante o conhecimento das preferências individuais, os indivíduos podem ser categorizados, de acordo com diferentes modelos, tipos psicológicos ou tipos de personalidade, os quais condicionam o seu estilo de aprendizagem.

Jung considera que, para além do complexo sistema de personalidade, um indivíduo apresenta atitudes (de introversão e extroversão) e funções (de pensamento, de sentimento, de sensação e de intuição). Para Jung a atitude introvertida orienta o indivíduo para o interior e subjectivo e a atitude extrovertida dirige a personalidade para o mundo exterior e objectivo. Segundo ele, todos temos ambas as atitudes, mas somente uma delas é dominante e consciente enquanto a outra é subordinada e inconsciente. As funções psicológicas propostas por Jung têm o seu papel na adaptação do indivíduo às situações da vida exterior ou interior. Pensamento é a função do conhecimento intelectual e da formação lógica de conclusões, sem interferência de valores pessoais. Sentimento é a função que está ligada a uma dimensão pessoal que valoriza pessoas e coisas. Sensação é a função da percepção obtida através dos órgãos sensoriais. Intuição é a função da percepção obtida por vias inconscientes que vai além dos factos, sentimentos e ideias, procurando o significado intrínseco das coisas e suas

possibilidades. As funções pensamento e sentimento são chamadas de racionais ou judicativas, por usarem a razão, o juízo, a abstracção e a generalização. As funções sensação e intuição são denominadas de irracionais por estarem baseadas na percepção, que se dirige para o que acontece, sem escolha judiciosa. Todos os indivíduos possuem as quatro funções, mas, devido à diversidade de circunstâncias, elas não são todas desenvolvidas da mesma forma.

Na segunda metade do século XX, Isabel Briggs Myers e a sua mãe Katherine Cook Briggs, retomaram a obra de Jung à qual adicionaram mais uma categoria que se prende com a forma dos indivíduos viverem e lidarem com o mundo exterior: a escolha entre uma atitude perceptiva e uma atitude julgadora.

"Embora as pessoas tenham que naturalmente usar tanto a percepção quanto o julgamento, estas não podem ser usadas ao mesmo tempo. [...] Existe um momento para perceber e outro momento para julgar, e muitas vezes qualquer dessas atitudes pode ser pertinente. A maioria das pessoas acha uma atitude mais confortável que a outra, ficando mais à vontade com ela e usando-a o mais frequentemente possível ao lidar com o mundo exterior" (Myers e Myers, 1980).

O instrumento utilizado para a classificação dos indivíduos, desenvolvido por estas autoras, é chamado *Myers-Briggs Type Indicator* – MBTI (Myers e McCaulley, 1985) e nele um indivíduo é qualificado em quatro categorias, de acordo com as seguintes dimensões: Extrovertido/Introvertido, Sensorial/Intuitivo, Reflexivo/Sentimental e Julgador/Perceptivo. Estas categorias são obtidas com base nas afirmações exibidas na tabela 3.1.

Os indivíduos **extrovertidos** geralmente focam-se no mundo externo, tendo mais facilidade para aprender quando as experiências e actividades precedem os conceitos e ideias. Apesar de poderem lidar eficientemente com abstracções, têm mais êxito trabalhando exteriormente, na acção. Necessitam de comunicar, procurando grupos de trabalho para socializar. Gostam de trabalhar rapidamente, de forma a apresentar os resultados o quanto antes, tendo por vezes dificuldade em entender uma ideia subjacente, pelo facto de não se deterem o tempo suficiente na tarefa em questão, apresentando a tendência para uma certa superficialidade intelectual. São normalmente mais verbais que os introvertidos.

Os indivíduos **introvertidos** apresentam maior facilidade para aprender quando os conceitos são explicados antes que se exijam experiências e resultados práticos. Concentram-se no mundo interno ou das ideias, preferindo trabalhar sozinhos, precisando de tempo para estudar e classificar uma nova situação. Uma vantagem dos introvertidos é a sua capacidade de concentração, o que os faz ignorar muitos dos estímulos exteriores que podem desviar a

sua atenção. Tendem a mostrar apenas as suas conclusões, sem grandes detalhes. Essa brevidade de comunicação faz com que sejam frequentemente encarados como tímidos.

E - Extroversão	**I – Introversão**
O interesse do indivíduo centra-se principalmente:	
No mundo exterior – acções, objectos ou pessoas.	No mundo interior – conceitos ou ideias.
S - Sensação	**I – Intuição**
O indivíduo percebe melhor:	
Os factos práticos e situações concretas e imediatas.	As possibilidades, relações e significados das coisas.
P - Pensamento	**S – Sentimento**
O indivíduo prefere fazer julgamentos ou tomar decisões:	
Objectivas e impessoais.	Subjectivas e pessoais.
J - Julgamento	**P – Percepção**
A pessoa prefere viver, maioritariamente:	
De uma forma planeada e ordenada, com uma atitude controladora.	De uma forma espontânea e flexível, com uma atitude compreensiva.

Tabela 3.1: Classificação das categorias do modelo de Myers-Briggs

Apesar dos indivíduos classificados como **sensoriais** poderem apreciar a ordem lógica e uma certa harmonia nas coisas em que se envolvem, a principal motivação para a aprendizagem é a natureza prática das coisas e a sua utilidade. Este tipo de indivíduos gosta de se focar em factos pelo que, muito provavelmente, ficarão perdidos quando o professor omite etapas nas explicações e instruções, deixando grandes lacunas para o aluno completar. Os estudantes classificados nesta categoria obtêm melhores resultados quando os seus sentidos estão completamente empenhados e embutidos na tarefa a realizar.

Os tipos classificados como **intuitivos** necessitam, acima de tudo, de inspiração. Para que o seu trabalho seja produtivo precisam de estar completamente envolvidos na tarefa a realizar, recheados de ideias e planos intrigantes. A rotina aborrece-os rapidamente. Frequentemente recorrem aos devaneios e elementos distractivos alheios à tarefa ou subestimam o professor sempre que este não é suficientemente persuasivo e interessante. Quando encontram algo diferente que lhes dê alento real são muito produtivos. Quando inspirados, são os mais inovadores.

Os indivíduos **reflexivos** tendem a tomar decisões com base em regras, sendo atraídos por materiais organizados logicamente; prosperam em coisas que requerem análise e apresentam dificuldades na aprendizagem de coisas que não se encaixem de forma lógica nos seus sistemas mentais. Geralmente são habilidosos e criativos no que concerne à resolução de problemas. Respondem melhor a professores bem organizados. Caso não encontrem uma ordenação lógica no material a estudar ou no professor, não conseguem direccionar as suas energias e esforços para as tarefas de aprendizagem.

Os indivíduos **sentimentais** tendem a tomar decisões com base em emoções e sentimentos. Normalmente apresentam duas preocupações fundamentais, relativas ao envolvimento com os professores e à importância da matéria a estudar. Caso consigam estabelecer um vínculo com o professor, conseguindo que este lhes dê atenção e descubram a pertinência da matéria a aprender, apresentam o seu melhor desempenho. Nestas situações o professor pode conduzir este tipo de estudante a desempenhar até as tarefas escolares que, à partida, pareciam não lhe interessar. Com ambas as condições ausentes, estes indivíduos perdem a sua principal motivação e quaisquer tentativas para os envolver terão grande probabilidade de resultar em fracasso.

Os indivíduos **julgadores** conseguem melhores resultados quando podem planear e controlar as suas tarefas e obedecer ao que foi agendado. Estabelecem o que deve ser feito não só para si, mas também para os que os rodeiam. Apreciam tarefas sistemáticas, disciplinadas, com funções bem definidas e estruturadas. Quando decidem fazer algo, fazem-no efectivamente, apresentando a tendência para completar os trabalhos. Devido a esta característica são vistos como responsáveis e dignos de confiança.

Os indivíduos **perceptivos** tendem a ser espontâneos nas suas acções e adaptam-se com facilidade a novos ambientes ou circunstâncias. Apreciam tarefas flexíveis, abertas e adaptáveis aos interesses de momento. São geralmente vistos como curiosos e não muito organizados, devido à sua tendência para deixar decisões e opiniões em aberto, à espera que surjam novas informações. A sua enorme curiosidade pode levá-los a um extraordinário amontoado de informações, sem que cheguem a alguma conclusão, muitas vezes, mesmo que dela necessitem.

Na tabela 3.2 apresenta-se um resumo com as principais características correspondentes às categorias de indivíduos, segundo o modelo de Myers-Briggs.

Extrovertidos	**Introvertidos**
Experimentalistas. Preferem estímulos exteriores. Preferem trabalhar em equipa. Tendência para estudos rápidos e superficiais.	Teóricos. Preferem examinar a informação interiormente. Preferem trabalhar sozinhos. Tendência para estudos demorados e profundos.
Sensoriais	**Intuitivos**
Preferem utilizar os sentidos: visão, audição, fala. Preferem informação concreta. Preferem coisas de natureza prática. Gostam de repetição e ordem lógica e sequencial das coisas em que se envolvem.	Preferem utilizar o pensamento: especular e imaginar. Preferem informação abstracta. Preferem coisas de natureza teórica. Gostam de coisas inovadoras e não apreciam a repetição.
Reflexivos	**Sentimentais**
Tomam decisões com base na razão. Precisam de ver ordem e lógica nos assuntos a aprender, para obterem desempenho nas suas tarefas.	Tomam decisões com base nos sentimentos e emoções. Precisam de se envolver com pessoas (alunos e professores) para obterem desempenho nas suas tarefas.
Julgadores	**Perceptivos**
Gostam de fazer coisas planeadas, passo-a-passo, com caminhos lógicos e estruturados. Apreciam a repetição. São apáticos/passivos.	Gostam de fazer coisas criativas, de forma não linear, seguindo caminhos aparentemente desorganizados e desconexos. Apreciam a inovação e flexibilidade. São curiosos.

Tabela 3.2: Caracterização das categorias do modelo de Myers-Briggs

Combinando-se as categorias, segundo este modelo, é possível obter 16 tipos possíveis de perfis. Por exemplo, um indivíduo classificado como ESRJ seria caracterizado como Extrovertido, Sensorial, Reflexivo e Julgador.

Apesar do modelo de Myers-Briggs evidenciar os tipos de personalidade, também é utilizado para medir os estilos de aprendizagem, uma vez que muitas categorias por ele definidas são baseadas em conceitos de natureza cognitiva.

3.3.2 Modelo de estilos de aprendizagem de Kolb

O modelo de aprendizagem de Kolb (Kolb, 1984) é, de acordo com o seu autor, designado de experiencial por duas razões principais. Uma por razões históricas, dadas as suas origens baseadas no trabalho de Kurt Lewin, Jean Piaget e Carl Jung. A outra, pelo facto de, no seu modelo, David Kolb enfatizar que a experiência exerce uma função primordial no processo de aprendizagem, uma vez que para ele a aprendizagem é um processo em que o conhecimento é criado através da transformação da experiência anterior ou conhecimento prévio (Kolb, 1984). Na perspectiva de Kolb um dos factores responsáveis pela falta de êxito na aprendizagem é a incapacidade de aprender com as experiências.

O modelo da teoria da aprendizagem experiencial foi proposto com uma perspectiva holística da aprendizagem que combina experiência, percepção, cognição e comportamento (Kolb, 1984). A essência do modelo consiste na descrição do ciclo de aprendizagem, ou seja, na descrição de como é que o indivíduo gera, a partir da sua experiência, os conceitos que guiarão o seu comportamento em novas situações, bem como a forma como o indivíduo modifica esses conceitos com a finalidade de melhorar a sua aprendizagem. Para tal, este modelo define um ciclo de aprendizagem repetitivo, formado por quatro fases consecutivas: Experiência Concreta (EC), Observação Reflexiva (OR), Conceptualização Abstracta (CA) e Experimentação Activa (EA), conforme ilustra a figura 3.1.

Na primeira fase do ciclo de aprendizagem, EC, ocorre o envolvimento com experiências concretas como ver, ouvir e tocar. Uma forma de promover este estado em contexto de sala de aula consiste em apresentar ferramentas e instrumentos que os alunos possam explorar e manipular.

A seguir, na fase denominada OR, inicia-se a observação e reflexão das experiências. Nesta fase o professor poderá compelir os alunos a reflectir sobre as experiências anteriores e compará-las com a experiência actual e, caso seja necessário, moldá-la de acordo com os seus padrões anteriores.

Na fase CA ocorre a integração e transformação das observações e reflexões em teorias e conceitos. O professor pode, em contexto de sala de aula ajudar a promover esta fase através da introdução de novos conceitos-chave, vocabulário e diagramas relacionados com o tema em questão.

Por último, as teorias são utilizadas para realizar tomadas de decisão e resolução de problemas na fase EA. Uma forma de o professor ajudar o aluno a consubstanciar esta fase consiste em ajudá-lo a testar e verificar o novo conceito que lhe é apresentado.

De acordo com este modelo, a aprendizagem pode ter início em qualquer fase do ciclo, mas o desejável é que o indivíduo consiga estabelecer ligações e percorrer todos os estágios para que a aprendizagem seja eficaz. Porém, cada indivíduo, dependendo das suas aptidões, pode conseguir melhor desempenho em determinada fase específica.

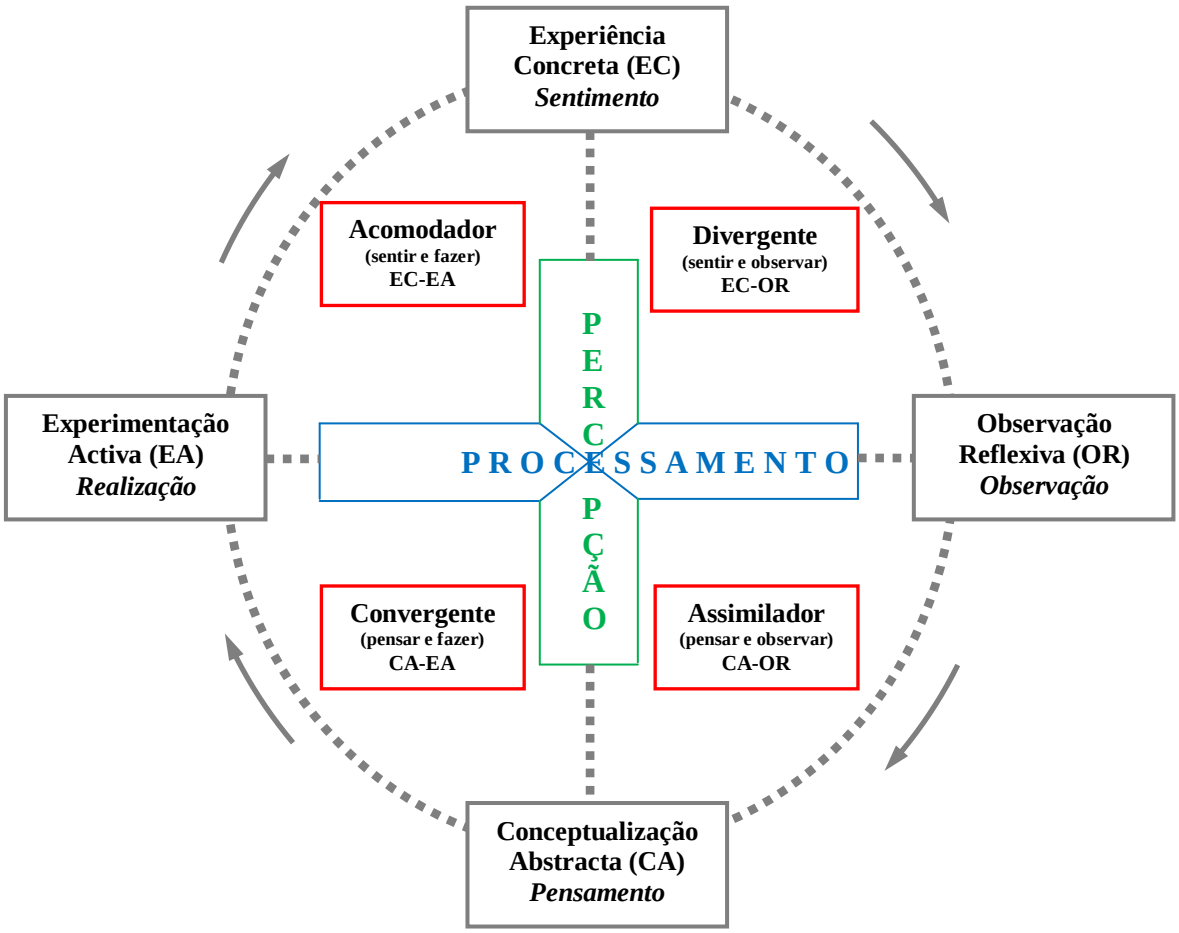


Figura 3.1: Ciclo da aprendizagem experiencial de Kolb

Uma análise mais profunda do modelo de David Kolb revela que, no processo de aprendizagem, há duas dimensões básicas, nomeadamente:

- Percepção da informação – referente aos canais de recepção. Os indivíduos podem ter aptidões para a Experiência Concreta (ver, ouvir, tocar) ou aptidões para a Conceptualização Abstracta (recorrendo a conceitos mentais).

- Processamento da informação – relacionada com a maneira como a informação é processada. Os indivíduos podem sentir-se mais confortáveis com a Observação Reflexiva (pensando sobre as coisas) ou a Experimentação Activa (fazendo algo com a informação).

Kolb desenvolveu um instrumento, o *Learning Style Inventory* (LSI) (Kolb, 1985), que permite classificar os indivíduos em quatro perfis de estilos de aprendizagem, resultantes da combinação das duas dimensões referidas acima, nomeadamente:

- **Divergente** (concreto, reflexivo) – As características predominantes neste estilo são EC e OR. Os canais preferenciais de recepção das informações são a observação visual e em última instância as percepções subjectivas. As informações são processadas de forma reflexiva, sem necessidade de experiência activa. Os indivíduos chamados de divergentes conseguem perceber a informação em diferentes perspectivas e sistematizá-la num todo coerente. Os seus pontos fortes são a capacidade de imaginação e percepção de significados e valores. Os aprendizes divergentes preferem envolver-se com circunstâncias reais e gostam de trabalhar em grupo. Os sujeitos que apresentam uma forte componente divergente podem ter dificuldades nas tomadas de decisões quando lhe são apresentadas múltiplas escolhas. Porém, os indivíduos que carecem desta componente podem ter dificuldade em gerar ideias e reconhecer problemas.

- **Assimilador** (abstracto, reflexivo) – As habilidades predominantes neste estilo são CA e OR. Os indivíduos assimiladores percebem as informações através de modelos mentais e processam-nas de forma reflexiva, sem necessidade de experiência activa. São muito teóricos, relacionam-se bem com abstracções. Geralmente organizam as informações através de raciocínio indutivo. São chamados assimiladores porque analisam, organizam e assimilam partes das informações, agrupando-a num todo integrado. Em algumas situações no processo de aprendizagem preferem as ideias do que as pessoas, preferindo trabalhar sozinhos. Os indivíduos fortemente assimiladores, correm o risco de ficar no mundo das ideias e não conseguirem aplicá-las na prática, no mundo real. Em contrapartida, os sujeitos que apresentam uma componente fraca deste perfil tendem a não aprender com os seus erros e ter dificuldades em sistematizar.

- **Convergente** (abstracto, activo) – As habilidades dominantes neste estilo são CA e EA. Os indivíduos pertencentes a este grupo percebem a informação por meio da conceptualização abstracta e processam-na activamente. Integram teoria e prática: testam as informações, experimentam coisas, vêem como funcionam e aprendem fazendo. São chamados de convergentes pela inclinação que possuem para procurar soluções correctas para as suas reflexões. São rápidos na tomada de decisão e eficientes na resolução de problemas. Preferem trabalhar com objectos em detrimento de pessoas. Os indivíduos fortemente convergentes podem tomar

decisões erradas em determinadas situações, devido à sua tendência para resolver problemas com demasiada rapidez.

- **Acomodador** (concreto, activo) – As características predominantes neste estilo são EC e EA. Os indivíduos preferem aprendizagens com base na experimentação activa e na experiência concreta. São inventivos e geralmente aplicam o conhecimento adquirido para resoluções de problemas reais. Adequam-se bem a situações imprevistas e agem muitas vezes pelo instinto em vez de pela lógica. Gostam de trabalhar em equipa e geralmente são impacientes. São chamados acomodadores por terem facilidade em aplicar as experiências apreendidas em seu próprio benefício, usando as descobertas para mudar e fazer melhor.

Embora segundo o modelo cíclico de David Kolb toda a aprendizagem para ser eficaz deva envolver as quatro fases, cada indivíduo, dependendo das suas aptidões, pode conseguir melhor desempenho numa fase específica. A tabela 3.3 apresenta de forma sintetizada os quatro perfis de estilos de aprendizagem, segundo o modelo de David Kolb, as suas principais tendências e aptidões dominantes.

Estilo	Características
Divergente	**Aptidões dominantes: EC-OR**
	O seu ponto forte é a imaginação e produção de ideias.
	Sobressae em situações que exigem geração de ideias.
	É emotivo, interessa-se por pessoas (gosta de trabalhar em grupo).
	Tem interesses culturais amplos. Tende a especializar-se em artes.
	Característico dos indivíduos das áreas de humanidades.
Assimilador	**Aptidões dominantes: CA-OR**
	O seu ponto forte é a capacidade de criar modelos teóricos.
	Sobressae em raciocínio indutivo (assimilar observações desencontradas e transformá-las em explicações integradas).
	Menos interessados por pessoas e mais em conceitos abstratos (ideias).
	Tem mais interesses teóricos do que nas suas aplicações práticas.
	Característico de pessoas que trabalham em departamentos de pesquisa e planeamento (Ciências, economia e física).
Convergente	**Aptidões dominantes: CA-EA**
	O seu ponto forte é a aplicação prática de ideias abstractas, aprende testando os seus conceitos.
	Sobressae em raciocínio hipotético-dedutivo, através do qual se consegue concentrar em problemas específicos.
	Pefere lidar com "coisas" a lidar com pessoas.
	Tem interesses técnicos específicos.
	Característico de muitos engenheiros.
Acomodador	**Aptidões dominantes: EC-EA**
	O seu ponto forte é realizar coisas, executar planos e experiências.
	Sobressae em situações nas quais precise de se adaptar às circunstâncias imediatas e específicas.
	Gosta de estar com pessoas, embora seja, geralmente, impaciente e pressionador.
	Tem interesses por experiências novas e em assumir riscos.
	Caracteristico de indivíduos de áreas de negócio/comércio.

Tabela 3.3: Caracterização das categorias do modelo de Kolb

3.3.3 Modelo de estilos de aprendizagem de Felder-Silverman

O modelo de estilos de aprendizagem proposto por Felder e Silverman (1988), de base conceptual assente nos conceitos e ideias apresentadas por Carl Jung, categoriza um indivíduo relativamente às formas pelas quais prefere receber e processar a informação, atribuindo-lhe uma determinada classificação. Assim, define cinco dimensões: Sensorial/Intuitivo, Visual/Verbal, Activo/Reflexivo, Sequencial/Global e Indutivo/Dedutivo. As dimensões Sensorial/Intuitivo e Visual/Verbal referem-se aos mecanismos de percepção da informação. As dimensões Activo/Reflexivo e Sequencial/Global dizem respeito à maneira como a informação é processada e transformada em conhecimento. De acordo com este modelo, os estilos de aprendizagem de um indivíduo podem ser definidos em função das afirmações expressas na tabela 3.4.

Sensorial	**Intuitivo**
O indivíduo percebe melhor a informação:	
Externa – imagens, sons, sensações físicas.	*Interna* – possibilidades, intuições, palpites.
Visual	**Verbal**
O indivíduo percebe a informação mais eficazmente, de forma:	
Visual – figuras, diagramas, gráficos.	*Verbal* – palavras, sons.
Activo	**Reflexivo**
O indivíduo prefere processar a informação:	
Activamente – por meio do envolvimento em actividades físicas ou discussões.	*Reflexivamente* – por meio da introspecção.
Senquencial	**Global**
O indivíduo prefere processar a informação:	
Sequencialmente – de uma forma contínua, passo-a-passo.	*Globalmente* – em grandes saltos, holisticamente.
Indutivo	**Dedutivo**
O indivíduo sente-se mais confortável organizando a informação de forma:	
Indutiva – são apresentados factos e informações e inferidos os princípios básicos.	*Dedutiva* – os princípios são dados e as consequências e aplicações são deduzidas.

Tabela 3.4: Classificação das categorias do modelo de Felder-Silverman

De seguida, é caracterizada cada uma das categorias, que constituem este modelo.

3.3.3.1 Sensorial/Intuitivo

Na sua teoria dos tipos psicológicos, Carl Jung introduziu as modalidades sensorial e intuitiva para significar as duas formas que as pessoas utilizam habitualmente para perceber o mundo. A modalidade sensorial envolve observar, recolher dados a partir dos sentidos; a modalidade intuitiva envolve a percepção indirecta pelo inconsciente - especulação, imaginação, pressentimentos.

Os indivíduos **sensoriais** apreciam factos e dados, gostam de resolver problemas por métodos pré-estabelecidos, não gostam de complicações e novas situações. São mais meticulosos, têm facilidade em memorizar factos e preferem trabalhos práticos. Em contrapartida, os indivíduos **intuitivos** preferem descobrir possibilidades e relações entre as coisas, gostam de novidades, aborrecem-se com as repetições e normalmente apreendem novos conceitos com grande facilidade e entusiasmo.

3.3.3.2 Visual/Verbal

As pessoas recebem informação através de diferentes modalidades, nomeadamente: visual – através de figuras, diagramas ou símbolos; auditiva – através de sons ou palavras; cinestésica – através do gosto, toque ou cheiro, sendo este último aspecto de menor importância para o presente trabalho. Diversas pesquisas mostraram que a maioria das pessoas aprende mais eficazmente através de uma das três modalidades referidas e apresenta tendência para "perder" a informação que lhes é apresentada através de uma das outras duas modalidades (Dunn e Dunn, 1978; Bandler e Grinder, 1979; Barbe et al., 1979; Barbe e Milone, 1981; Waldheim, 1987).

Os indivíduos caracterizados como **visuais** relembram com facilidade o que viram e aprendem melhor visualizando figuras, diagramas, fluxogramas, filmes ou demonstrações. Provavelmente esquecer-se-ão de algo que lhes é comunicado apenas de forma verbal. Os **verbais** conseguem tirar maior proveito das palavras e explicações escritas e faladas, em detrimento de demonstrações visuais.

3.3.3.3 Activo/Reflexivo

Os complexos processos mentais pelos quais a informação percebida é convertida em conhecimento podem ser agrupados em duas categorias: a experimentação activa e a observação reflexiva. A experimentação implica fazer algo com a informação no mundo externo – discuti-la, explicá-la ou testá-la de alguma forma, a observação reflexiva envolve examinar e manipular a informação introspectivamente.

Os indivíduos classificados como **activos** preferem discutir a informação, explicar aos outros e aplicar na prática os conceitos aprendidos. Gostam de trabalhar em grupo e de fazer experiências para entender os assuntos. Por outro lado, os indivíduos classificados como **reflexivos** preferem pensar primeiro sobre a informação e entender o tema para depois realizar experiências e geralmente preferem trabalhar sozinhos ou quando muito com uma pessoa.

Os activos tendem a ser experimentalistas, os reflexivos tendem a ser teóricos. Numa primeira análise, parece existir uma sobreposição considerável entre os activos e os sensoriais, pois ambos apreciam estar envolvidos no mundo externo dos fenómenos, e entre os reflexivos e os intuitivos, na medida em que ambos preferem o mundo interno da abstracção. Porém, as categorias são independentes. O sensorial selecciona preferencialmente a informação disponível no mundo externo mas pode processá-la activa ou reflexivamente, no último caso postulando explicações ou interpretações, construindo analogias ou formulando modelos. Similarmente, o intuitivo selecciona a informação gerada internamente mas pode processá-la reflexiva ou activamente, no último caso construindo uma experiência para testar a ideia ou experimentando-a com um colega.

3.3.3.4 Sequencial/Global

Os indivíduos sequenciais tendem a aprender de forma linear e em etapas logicamente sistematizadas. Tentam seguir caminhos lógicos para encontrar soluções. Geralmente não têm problemas com a forma tradicional de aprendizagem porque a maioria dos livros e professores apresentam a matéria de forma sequencial. Aprendem melhor quando o material é apresentado através de uma progressão constante de complexidade e dificuldade.

Os **globais** tendem a aprender em grandes saltos, assimilando o conteúdo aparentemente de forma aleatória, sem ver as conexões, para então, quase repentinamente compreenderem o todo. Normalmente, obtêm melhores desempenhos ao "saltarem" directamente para materiais mais complexos e difíceis. São hábeis a resolver problemas complexos com rapidez. Quando um conteúdo lhes é apresentado pela primeira vez, podem ficar um pouco "desnorteados", mas depois de fazer algumas conexões com outras disciplinas ou assuntos do seu conhecimento podem atingir a compreensão quase que instantaneamente. Quando encontram uma resposta para um problema, muitas vezes não sabem explicar com pormenor o caminho percorrido para encontrar a solução.

Os **sequenciais** conseguem trabalhar com assuntos mesmo quando os compreendem apenas parcial ou superficialmente enquanto que os globais podem ter grande dificuldade em

o fazer. Normalmente, os sequenciais são bons em pensamento convergente e em análise enquanto que os globais podem ser melhores em pensamento divergente e em síntese.

3.3.3.5 Indutivo/Dedutivo

A indução é uma progressão do raciocínio que parte dos detalhes (observações, medidas, dados) para as generalidades (regras, leis, teorias). A dedução prossegue no sentido oposto. Na indução inferem-se princípios; na dedução deduzem-se consequências. A indução é o estilo de aprendizagem natural dos seres humanos, basta reparar que quando nascemos não vimos "equipados" com um conjunto de princípios gerais, mas ao observarmos o mundo à nossa volta aprendemos a extrair inferências. Assim, a maioria do que aprendemos por nós próprios tem origem numa situação ou problema real que necessita de ser resolvido, e não num princípio geral.

Os **indutivos** organizam as informações partindo de uma sequência de raciocínio particular, progredindo para o geral, inferem princípios. Os **dedutivos** organizam as informações de modo a que as soluções sejam consequências de uma ideia geral, deduzem consequências. É considerado o estilo predominante de ensino na maioria dos cursos técnicos de ensino superior. Os professores partem de uma teoria e trabalham com os alunos até chegarem às suas aplicações.

3.4 Index of Learning Styles

Nesta secção apresenta-se o ILS (*Index of Learning Styles*) (Soloman e Felder), proposto para identificar os estilos de aprendizagem, de acordo com o modelo de Felder-Silverman. Esta apresentação justifica-se por ter sido este o modelo e o instrumento usado nos nossos estudos.

Richard Felder e Barbara Soloman desenvolveram em 1991 a primeira versão do ILS, o qual apresentava 28 questões, tendo por objectivo identificar as preferências de aprendizagem de acordo com o modelo de Felder-Silverman. Em 1996 foi publicada uma nova versão do ILS com várias alterações. Actualmente o ILS (Anexo G) é composto por 44 perguntas de escolha obrigatória a) ou b), 11 para cada uma das quatro dimensões Sensorial/Intuitivo, Visual/Verbal, Activo/Reflexivo e Sequencial/Global. Apesar de Felder apresentar a dimensão Indutivo/Dedutivo ela não é avaliada no seu instrumento, pelo facto de o autor considerar que todo o processo de ensino deva ser realizado de forma indutiva. A classificação de um indivíduo numa ou noutra categoria pode ser forte (9 a 11), moderada (5 a 7) ou fraca (1 a 3). Um indivíduo cuja categoria seja classificada como fraca apresenta preferência sensivelmente

indiferente por qualquer das categorias dessa dimensão. Uma classificação moderada significa que o indivíduo tem alguma preferência por determinado pólo dessa dimensão. Os indivíduos com uma pontuação elevada indicam uma forte preferência por uma determinada categoria dessa dimensão apresentando dificuldade quando os materiais de aprendizagem lhe são apresentados através de técnicas opostas.

A figura 3.2 mostra um exemplo de um resultado gerado após a resposta de um aluno ao teste ILS.

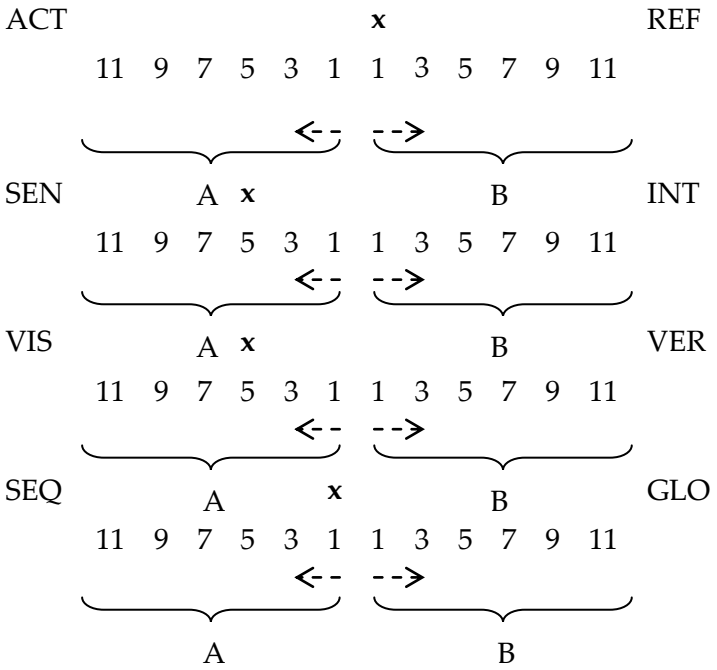


Figura 3.2: Exemplificação do resultado de um teste ILS

O ILS abrange em cada uma das quatro dimensões, duas categorias ou estilos opostos de aprendizagem: activo (ACT) ou reflexivo (REF); sensorial (SEN) ou intuitivo (INT); visual (VIS) ou verbal (VRB) e sequencial (SEQ) ou global (GLO). As respostas às questões do instrumento fornecem, para cada uma das quatro dimensões, uma pontuação que corresponde à categoria abrangida pela dimensão, indicando, desta forma, o estilo predominante ou preferido pelo respondente. A título de exemplo, o aluno cujo resultado se encontra na Figura 3.2 é fracamente reflexivo (REF=1), moderadamente sensorial (SEN=5), moderadamente visual (VIS=5) e fracamente sequencial (SEQ=1).

O ILS está disponível na *world wide web* gratuitamente, através do endereço, http://www.engr.ncsu/learningstyles/ilsweb.html.

3.5 Estudos referidos na Literatura

Diversos trabalhos relatam a utilização, no sistema educativo superior, de variados modelos de estilos de aprendizagem, com o intuito de proporcionar um ensino mais voltado para as preferências e interesses dos alunos e assim favorecer o sucesso escolar. Podemos referir, por exemplo, o estudo conduzido por Stice (1987), que apresenta a aplicação do modelo de aprendizagem de Kolb, o LSI, ao ensino e aprendizagem de equações diferenciais ordinárias, na disciplina de Controlo de Processos de um Curso de Engenharia Química. Outro exemplo é o do estudo conduzido por Montgomery (1995), com 143 alunos de Engenharia Química da *University of Michigan*, sobre as diferentes formas através das quais a multimédia pode ser utilizada, para atender aos diferentes perfis de aprendizagem dos alunos que frequentam a disciplina de Introdução à Engenharia Química. Neste estudo o autor utilizou o ILS de Felder e Soloman para determinar as preferências de aprendizagem. Os resultados mostraram que a maioria dos alunos eram sensoriais (57%), visuais (69%), activos (67%) e sequenciais (71%). Outro exemplo é o estudo conduzido por Rutz et al. (2000), na Faculdade de Engenharia da *University of Cincinnati* em cooperação com a *Wright State University*, utilizando o LSI de Kolb e o MBTI para determinar a existência de correlações entre os estilos preferenciais de aprendizagem dos indivíduos e as tecnologias educativas usadas. Os autores referem que apesar das engenharias atraírem uma grande variedade de tipos MBTI (ISFJ (5,3%), INFJ (3,2%), INTJ (5,8%), ISTP (6,3%), ISFP(4,2%), INFP (6,3%), ESTP (3,7%), ESFP (1,6%), ENTP (4,2%), ESTJ (3,7%), ESFJ (3,2%), ENFJ (4,7%), ENTJ (6,2%)), os tipos ISTJ(15,3%), ENFP(13,7%) e INTP(12,6%) predominam. Relativamente à aplicação do LSI de Kolb, foram encontrados os seguintes tipos: Convergente (43,7%), Assimilador (35,4%), Acomodador (11,5%) e Divergente (9,4%). Os autores não obtiveram correlações entre a utilização dos diferentes tipos de tecnologias educativas (vídeos interactivos ou materiais educativos baseados na *world wide web*) e os diferentes perfis de estilos de aprendizagem, em qualquer dos modelos. No entanto, os resultados indiciaram que os estudantes tinham preferências por usar os estilos convergente ou assimilador quando confrontados com os modelos de aulas tradicionais.

Têm também sido realizados estudos para caracterizar os alunos das engenharias em geral**.** Oito escolas americanas de engenharia formaram um consórcio (*ASEE MBTI Engineering Consortium*) com o objectivo de estudar os efeitos das diferenças dos tipos de personalidade no desenvolvimento educativo e profissional dos seus estudantes. Os autores do estudo (O'Brien et al., 1998) aplicaram o MBTI a 3718 alunos (do 1º ano) dos cursos de Engenharia Civil, Engenharia Eléctrica, Engenharia Mecânica e Engenharia de Produção. Os resultados revelaram que os estudantes eram marcadamente reflexivos (74%) e julgadores (61%); que os

do sexo masculino eram mais introvertidos (56%); que os estudantes estavam quase igualmente divididos entre os tipos sensoriais (53%) e intuitivos (47%).

Felder (1993, 1995 e 1996), relata a utilização do seu ILS para diagnosticar os estilos de aprendizagem de alunos de engenharia da *North Carolina State University*. Os resultados dos seus estudos evidenciaram que os estilos de aprendizagem de grande parte dos alunos de engenharia e os estilos de ensino da maioria dos professores eram incompatíveis em várias dimensões. A maioria dos alunos eram sensoriais, visuais, activos e sequenciais, enquanto a maioria dos seus professores promovia aulas de acordo com os estilos intuitivos, verbais, reflexivos e globais.

Rosati (1996) descreve os resultados das respostas ao ILS de Felder e Soloman aplicado a dois grupos de estudantes (1º e 4º anos) de engenharia da *University of Western Ontario* e a um grupo de professores que participaram numa sessão sobre estilos de aprendizagem na "*ASEE 1995 Annual Conference*" (Rosati, 1995). Os alunos do primeiro e quarto ano demonstraram uma preferência significativamente maior para a aprendizagem activa do que os professores. Surpreendentemente a preferência para a aprendizagem activa era mais forte no 4º ano do que no primeiro, talvez devido à maior ênfase, nos últimos anos, de trabalhos práticos e de estudo em grupo. Ambos os grupos de estudantes expressaram uma preferência sensorial significativamente maior do que os professores. Todos os três grupos relataram uma preferência de aprendizagem visual, mas a preferência dos professores era significativamente mais forte (mais visual) do que a dos alunos. A engenharia é uma disciplina visual, desta forma as preferências visuais eram esperadas. Apesar das suas preferências de aprendizagem visual, os professores, revelaram-se consideravelmente verbais nas suas formas de ensino, provavelmente porque tais apresentações são mais fáceis de realizar. Os estudantes do 1º ano eram significativamente mais sequenciais do que os estudantes do 4º ano, e estes últimos também mais sequenciais do que os professores. Os professores eram mais intensamente reflexivos, intuitivos e visuais nas suas preferências de aprendizagem do que os alunos do 1º ano ou os do 4º ano. Quer os professores, quer os alunos do 4º ano expressaram uma preferência de aprendizagem mais global do que os alunos do 1º ano. No entanto, os estilos de ensino da maioria dos professores tendiam a enfatizar os procedimentos reflexivos, intuitivos, verbais e sequenciais. Os autores do estudo referem ainda que os professores de engenharia devem reconhecer que as suas aulas incluem todos os tipos de aprendizes, pelo que uma instrução eficaz deve apelar a cada estilo de aprendizagem através de actividades equilibradas.

Sloan (1998), num estudo realizado em oito escolas de engenharia (*University of Alabama*, *Clemson University*, *Cleveland State University*, *Colorado School of Mines*, *The Cooper Union*, *University of Floriday*, *University* of *Houston* e *Indiana University/Purdue University at*

Indianapolis) observou também a incompatibilidade de estilos existente entre alunos e professores, através da aplicação do instrumento MBTI. Nesse estudo, constatou que mais de 77% dos professores eram intuitivos, enfatizavam conceitos e utilizavam poucos exemplos. Pelo contrário, quase metade dos estudantes de engenharia (46%) eram sensoriais, preferindo múltiplos exemplos para a aprendizagem de conceitos.

Um novo estudo conduzido por Rosati (1999), incluindo um grupo de 858 alunos de engenharia da *University of Western Ontario*, ao qual foi aplicado o ILS de Felder e Soloman confirmou a tendência verificada noutros estudos, nomeadamente que a maioria dos alunos apresentava uma preferência sensorial (59%), visual (80%), activa (69%) e sequencial (67%).

Num estudo conduzido por Allen e Mourtos (2000), na Faculdade de Engenharia da *San José State University*, foi aplicado o ILS de Felder e Soloman, para categorizar os estilos de aprendizagem de 319 estudantes de engenharia. Os resultados demonstraram mais uma vez que estes estudantes eram maioritariamente sensoriais, visuais, activos e sequenciais.

Num estudo conduzido por Buxeda et al. (2001), na *Universidad de Puerto Rico* foi realizada uma revisão da disciplina de Reconhecimento de Padrões. Essa revisão foi iniciada pela caracterização do perfil de aprendizagem de cada aluno usando o ILS de Felder e Soloman. Mais uma vez, foi demonstrada a predominância dos aprendizes sensoriais, visuais, activos e sequenciais. A determinação destes perfis possibilitou a definição de uma estrutura para o planeamento e concepção de actividades de forma a potenciar as preferências de aprendizagem da população estudada. Porém, estas modificações requereram, consequentemente, a modificação dos procedimentos de avaliação. Destes destacam-se a existência de um portfólio para cada aluno, bem como o enriquecimento dos exames e trabalhos de casa com ferramentas para controlar o desempenho dos alunos e as suas competências orais e escritas.

Também Zywno (2002) realizou várias experiências com 352 alunos do curso de Engenharia Electrotécnica e de Computadores da *Ryerson University*, em Toronto, entre 1999 e 2002, utilizando o ILS de Felder e Soloman. As distribuições encontradas foram similares em todos os anos do estudo, com a maioria dos estudantes de engenharia identificados como aprendizes sensoriais (68%), visuais (87,4%), activos (57,9%) e sequenciais (62,8%). De notar que as percentagens apresentadas dizem respeito a valores médios do conjunto dos anos considerados. Por outro lado, este autor realizou outra pesquisa (Zywno, 2003) sobre as preferências de aprendizagem e estratégias de ensino utilizadas por 48 professores de engenharia que revelou que os docentes eram predominantemente intuitivos (58,3%), visuais (93,6%), reflexivos (62,5%) e globais (64,6%). Sendo assim, à excepção da preferência visual, o perfil dos docentes era o oposto do perfil de grande parte dos alunos. Neste estudo, o autor

concluiu que os estilos intuitivo e reflexivo da maioria dos docentes eram compatíveis com os métodos tradicionalmente utilizados nas aulas teóricas, que enfatizam a exposição teórica, mas promovem pouco *feedback* e não encorajam as interacções com os estudantes. Como tal, esses métodos não atendem às necessidades da maioria dos estudantes das engenharias que são, predominantemente, sensoriais e activos. Apesar da maioria dos docentes ter forte preferência pela aprendizagem visual, a pesquisa também demonstrou que os professores usavam poucas tecnologias educativas para apelar ao estímulo visual, como por exemplo as simulações computacionais. Isto revelou-se particularmente evidente no caso dos membros mais jovens do corpo docente. Paradoxalmente, apesar da proliferação da tecnologia, este estudo revelou que os docentes mais jovens eram os que menos as usavam nas suas práticas pedagógicas. Os docentes mais jovens foram também aqueles que menos participaram na investigação, o que, segundo estes autores, sugeria pouco interesse e a baixa prioridade por eles concedida aos problemas educativos.

Kuri (2004) realizou um estudo com 840 alunos matriculados nos cursos das Engenharias Civil, Mecânica, Produção e Electrotécnica da Universidade Federal de São Carlos. Aplicou o ILS de Felder e Soloman aos estudantes da amostra, os quais se encontravam distribuídos pelos 1º, 3º e 5º anos dos respectivos cursos. Após a análise de resultados, por ano do curso, constatou-se que no 5º ano havia uma maior percentagem de estudantes activos do que no 1º ano. Verificou-se também que nos 3º e 5º anos a quantidade de alunos que preferia estudar em grupo era superior à do 1º ano; no 3º ano o número de alunos que preferia experimentar as coisas e pensar sobre como fazê-las era maior do que no 1º. Também se verificou que no 5º ano o número de alunos que preferia realizar actividades fora da parte lectiva era maior. Kuri verificou que os estilos de aprendizagem dominantes nos cursos de engenharia investigados eram sensorial, visual, activo e global. Os aspectos mais marcantes foram o facto de todos os cursos apresentarem estudantes moderada ou fortemente visuais e o número de alunos sequenciais nos primeiros anos ser superior aos do 5º ano.

Zualkernan et al. (2006) realizaram um estudo comparativo entre a *University of Minnesota Duluth* (UMD), nos Estados Unidos da América e a *American University of Sharjah* (AUS), nos Emirados Árabes Unidos, Dubai. A ideia era verificar se os estilos de aprendizagem tinham alguma correlação com as diferentes culturas (ocidental e oriental). O estudo foi realizado em Outubro de 2004, envolvendo estudantes de Engenharia Electrotécnica e Ciências da Computação. O estudo concluiu que a maioria dos estudantes das duas universidades eram sensoriais, visuais e sequenciais. Relativamente ao pólo activo, o estudo mostrou uma pequena diferença, 51% dos estudantes da UMD eram activos versus 46% dos estudantes da AUS. Tendo como base a amostra, verificou-se que não existiam diferenças significativas de estilos de aprendizagem entre os estudantes ocidentais e orientais.

Nesta pesquisa interessou-nos fundamentalmente verificar a existência de estudos que correlacionassem os estilos de aprendizagem com o desempenho a programação. Relativamente a este aspecto, citemos por exemplo, o estudo conduzido por Chamillard e Karolik (1999) com 877 estudantes da *United States Air Force Academy*, na disciplina de Introdução à Computação no ano de 1997/1998, onde se verificou que os alunos classificados como reflexivos obtiveram melhor desempenho nessa disciplina, relativamente aos activos. Também Byrne e Lyons (2001) conduziram um estudo com a finalidade de examinar a relação entre os resultados dos alunos à primeira disciplina de programação e a predisposição de determinados factores tais como o género, a experiência prévia em programação, estilos de aprendizagem e outros factores académicos. Relativamente à questão dos estilos de aprendizagem os 91 alunos foram submetidos ao teste LSI de Kolb. O maior grupo de estudantes (37%) foi categorizado como convergente. Apesar de não se encontrarem correlações estatísticas entre o desempenho a programação e determinado estilo de aprendizagem, os alunos caracterizados como convergentes apresentaram resultados globais ligeiramente superiores aos restantes. De notar que, os indivíduos convergentes combinam a conceptualização abstracta e a experimentação activa. Os seus pontos fortes residem na resolução de problemas, tomadas de decisões e raciocínio dedutivo, aspectos considerados desejáveis em disciplinas de programação.

Thomas et al. (2002) realizaram um estudo com 107 estudantes matriculados na disciplina de Introdução à Computação, *University of Wales,* em *Aberystwyth*, no Reino Unido, e constataram que a maioria dos estudantes eram activos (54%) e visuais (77%). No entanto a percentagem de sequenciais (50,5%) e sensoriais (50,5%) foi neste caso ligeiramente diferente dos padrões habitualmente encontrados para alunos de Engenharia. Estes estudos evidenciaram, de forma estatisticamente significativa, que os alunos reflexivos e verbais foram aqueles que apresentaram melhor desempenho quando avaliados num exame de programação. A mesma tendência foi verificada no estudo conduzido por Allert (2004).

Como resultado do conhecimento dos estilos de aprendizagem e consequentes aplicações de diversos instrumentos, têm surgido recomendações para práticas de ensino consonantes e mais eficazes. Um estudo conduzido por Ayre e Nafalski (2000) na *University of South Australia* teve como resultado a proposta de reformulação dos métodos de ensino e avaliação, de forma a atender à diversidade de estilos de aprendizagem encontrados. As iniciativas tomadas tiveram como objectivo tornar o currículo de Engenharia Electrotécnica mais inclusivo. Os autores consideraram os seguintes aspectos como prioritários para atender às necessidades de um público heterogéneo: focar-se nas necessidades dos alunos; ser acessível e disponível; desenvolver um relacionamento de abertura e respeito; negociar os métodos de ensino e de avaliação; fornecer *feedback* rápido e frequente; utilizar metodologias

de ensino e aprendizagem inovadoras e flexíveis; rever constantemente a eficácia dos métodos de ensino; assegurar a relevância dos conteúdos ensinados para as práticas profissionais.

Chang e Chang (2000) fazem também recomendações para melhorar o planeamento, ensino e avaliação dos alunos de um curso de Engenharia Electrotécnica. Os conselhos deixados por estes autores são: reestruturar o currículo para suportar e desafiar cada tipo de estilo de aprendizagem; incluir actividades para os alunos desenvolverem outros estilos diferentes dos seus; desenvolver projectos de grupo incluindo elementos com uma combinação de tipos opostos; formular um sistema para avaliar compreensões teóricas e aplicações práticas. Estes autores referem que o planeamento dos cursos com foco nestas ideias não só permite que os alunos reforcem as suas habilidades mas também desenvolvam competências nas áreas percepcionadas como sendo as suas fraquezas. De acordo com estes autores, existem factores que condicionam os resultados da aprendizagem e sobre os quais os educadores não têm possibilidade de exercer controlo, como a habilidade inata e a formação. No entanto, podem utilizar instrumentos como o MBTI para estruturar e desenvolver os currículos de forma a compatibilizar os estilos de aprendizagem e de ensino.

Watkins et al. (2003) implementaram um projecto baseado na *world wide web* que objectivava analisar o impacto do ambiente educativo na aprendizagem, numa disciplina dos cursos de Engenharia Electrotécnica e Física da *University of Missouri-Rolla*, tendo em conta as preferências individuais. Cada aprendiz podia dimensionar a experiência de aprendizagem em relação à profundidade do conteúdo, ordem de apresentação e repetição dos assuntos. Assim, estes autores desenvolveram módulos tutoriais assíncronos em duas versões — uma para contemplar estudantes sequenciais e outra para estudantes globais. Estes autores utilizaram quatro instrumentos: o ILS de Felder e Soloman, testes para avaliar a aprendizagem depois de cada lição específica, provas mensais e no final da disciplina e inquéritos sobre a satisfação do estudante no final de cada sessão. Os resultados indicaram um alto nível de satisfação entre os estudantes; para além dos estudantes com estilo global e sequencial de aprendizagem terem apresentado um melhor entendimento da matéria quando utilizaram o modulo compatível com o seu estilo.

3.6 Sugestões de Estratégias de ensino/aprendizagem

Estamos convictos de que a tomada de consciência da coexistência de vários perfis de aprendizagem em sala de aula, por parte de alunos e professores, permitirá a prática de estratégias, tendentes a contemplar a diversidade de preferências e consequentemente melhorar a aprendizagem. Que técnicas é que o professor deverá então utilizar para contemplar todos os estilos de aprendizagem? E os alunos, será que poderão contribuir para promover a sua aprendizagem, se aplicarem técnicas favoráveis às suas preferências? Finalizamos este capítulo com um conjunto de sugestões de ensino e aprendizagem, tendo como referência a utilização do modelo de Felder-Silverman e algumas recomendações encontradas na literatura (Felder, 1993; Felder, 1996; Felder e Brent, 2000; Felder et al., 2000; Felder et al., 2002; Felder e Brent, 2005).

3.6.1 Sensorial/Intuitivo

Um professor deve estar ciente que um estudante que prefira a abordagem intuitiva à percepção sensorial aprende melhor através de conteúdos abstractos (conceitos, teorias, fórmulas e símbolos) do que através de conteúdos concretos (factos, actividades observáveis). De forma a contemplar os dois pólos desta dimensão, o material apresentado aos alunos deve consistir numa mistura de informação concreta (factos, dados, fenómenos observáveis) e de conceitos abstractos (princípios, teorias, modelos matemáticos).

Outro aspecto fundamental a ter em atenção, relativamente a esta categoria, é o facto de os sensoriais serem geralmente meticulosos e apreciarem detalhes, mas poderem ser demorados nas suas tarefas. Pelo contrário os intuitivos são mais rápidos mas muitas vezes descuidados, ficando aborrecidos com demasiados detalhes e repetições. Face ao exposto, uma consideração importante consiste em prever cuidadosamente o tempo disponibilizado para a realização de actividades, por causa dos sensoriais. Os intuitivos, por outro lado, são mais propensos para a abstracção e sentem-se mais confortáveis com símbolos do que os sensoriais. Uma vez que as palavras são símbolos, traduzi-las no que representam é naturalmente fácil para os intuitivos, mas um esforço adicional para os sensoriais. Para além de gostarem de procedimentos detalhados e meticulosos, a lentidão acrescida dos sensoriais devido à necessidade de traduzir palavras coloca-os em desvantagem em testes cronometrados, podendo ter que ler as perguntas diversas vezes antes de começar a responder. Por outro lado, o tipo e dimensão das questões colocadas devem também ser equacionados, devido à impaciência dos intuitivos perante os detalhes (perguntas muito grandes, podem induzi-los a

começar a responder antes de as lerem completamente, levando-os a cometer erros desnecessários).

Uma vez que os alunos sensoriais relembram e compreendem melhor a informação se conseguirem ver como é que ela se interliga com o mundo real, se estes alunos estiverem perante um professor que enfatize conceitos abstractos e teóricos, poderão ter dificuldade. Sendo assim, estes alunos deverão questionar o professor sobre exemplos específicos dos conceitos apresentados, bem como pedir explicações sobre a aplicação desses conceitos na prática. Caso o professor não forneça exemplos suficientes o aluno deve procurar meios para os encontrar, fazendo pesquisas ou envolvendo-se em sessões de *brainstorming* com os colegas.

Se os alunos intuitivos se encontrarem perante situações que lhe são desfavoráveis, como as aulas de professores que enfatizam estratégias de memorização em detrimento da compreensão ou actividades muito repetitivas, podem ficar prejudicados. Estes alunos deverão ter a coragem de pedir aos professores interpretações ou teorias gerais que fundamentem e se liguem com os factos apresentados ou tentar por si próprios estabelecer essas ligações. Como já referido, estes alunos são também propensos a cometerem erros, devido à sua impaciência com os detalhes. Também, pelo facto de não gostarem de repetição, geralmente não verificam completamente as suas resoluções. Sugere-se a estes alunos, especialmente em situações de avaliação, que façam o esforço para ler as perguntas na sua totalidade, antes de começar a responder, bem como de verificar os resultados após as suas resoluções.

A maioria dos cursos de engenharia enfatiza conceitos em vez de factos, utilizam poucos exemplos e usam principalmente exposições e leituras (palavras, símbolos) como primeira aproximação para abordar os conteúdos, favorecendo assim os aprendizes intuitivos. Diversos estudos referidos anteriormente mostram que a maioria dos professores é ela própria intuitiva. Porém, a maioria dos estudantes de engenharia é sensorial, sugerindo uma má combinação de estilos de ensino/aprendizagem. De notar, no entanto, que ambos os perfis são essenciais à prática da engenharia. Muitas tarefas da engenharia requerem atenção a detalhes, experiências minuciosas, os principais marcos dos sensoriais; muitas outras tarefas requerem criatividade, habilidade teórica, e talento em suposições e conjecturas inspiradas, o que caracteriza os intuitivos. A tabela 3.5 apresenta um resumo dos principais aspectos a considerar face à dimensão Sensorial/Intuitiva dos estilos de aprendizagem.

Características	
Sensoriais	**Intuitivos**
Utilizam os sentidos: visão, audição, fala.	Utilizam o pensamento: especulam, imaginam.
Preferem informação concreta (factos, dados e fenómenos observáveis).	Preferem informação abstracta (conceitos, princípios, teorias e fórmulas matemáticas).
Preferem resolver problemas por métodos pré-estabelecidos.	Preferem descobrir novos métodos, possibilidades e relações entre as coisas.
Gostam de repetição, não apreciam surpresas ou novas situações.	Gostam de inovação, não apreciam repetições.
Meticulosos, pacientes com os detalhes.	Generalistas, impacientes com os detalhes.
Práticos.	Teóricos.
Actividades adequadas	
Promover actividades práticas.	Promover actividades teóricas.
Mostrar conexões do assunto com situações reais.	Dar tempo suficiente para os alunos desempenharem as suas tarefas.
	Colocar frequentemente questões para assegurar que o aluno compreendeu a tarefa.

Tabela 3.5: Caracterização dos Sensoriais/Intuitivos do modelo de Felder-Silverman

3.6.2 Visual/Verbal

Em geral, poucos professores teriam que modificar o que geralmente fazem de forma a contemplar os alunos verbais, nomeadamente em aulas teóricas e teórico-práticas. Porém, de modo a contemplar os estudantes visuais, deveriam intensificar o uso de diagramas, esquemas ou esboços para melhor ilustrar processos complexos. Sempre que possível deveriam recorrer a representações alternativas às verbais para a explicação do mesmo assunto, por exemplo, ao usarem funções matemáticas deveriam complementá-las por gráficos e, sempre que possível, deveriam introduzir filmes, simulações ou demonstrações de processos.

Caso um aluno visual se encontre perante um professor que enfatize a comunicação verbal, dever-se-á esforçar, por si próprio, por encontrar diagramas, esquemas, fluxogramas ou qualquer outro tipo de representação visual que melhor traduza a informação verbal apresentada. O aluno visual poderá perguntar ao professor se conhece ou tem disponíveis materiais visuais de apoio à disciplina. A preparação, pelo próprio aluno, de mapas conceptuais, a listagem de pontos ou conceitos-chave ou o destacar com determinada cor os elementos relacionados com determinado tópico, podem constituir importantes ferramentas de ajuda. Por outro lado, a escrita de resumos, através das suas próprias palavras, o trabalho

em grupo, ouvindo a explicação dos colegas e as suas tentativas de explicação, podem constituir aspectos importantíssimos para a aprendizagem dos alunos verbais.

A maioria dos alunos são visuais, mas o ensino praticado, em especial nas aulas teóricas, onde os conceitos estruturantes são introduzidos, é predominantemente verbal (conferências, palestras, seminários, métodos expositivos) ou uma representação visual da informação verbal (palavras e símbolos matemáticos escritos em textos, materiais de apoio, transparências, projecções, entre outros). Assim, verifica-se habitualmente uma má combinação dos estilos de ensino/aprendizagem existentes entre a modalidade preferida para recepção de informação na maioria dos alunos e a modalidade preferida de apresentação da maioria dos professores. A tabela 3.6 ilustra um quadro resumo com os principais aspectos a considerar face à dimensão Visual/Verbal dos estilos de aprendizagem.

Características	
Visuais	**Verbais**
Preferem informação visual (figuras, diagramas, fluxogramas, filmes e demonstrações).	Preferem informação verbal (palavras, explicações escritas e faladas).
Relembram com facilidade o que vêem.	Relembram com facilidade o que ouvem.
Exprimem-se melhor por figuras.	Exprimem-se melhor por palavras.
Actividades adequadas	
Usar diagramas, gráficos, esquemas ou esboços.	Usar textos, explicações verbais, fórmulas.

Tabela 3.6: Caracterização dos Visuais/Verbais do modelo de Felder-Silverman

3.6.3 Activo/Reflexivo

O professor deve alternar as apresentações com pausas ocasionais para reflexão (favorecendo os reflexivos) e breves discussões ou actividades de resolução de problemas (favorecendo os activos), e deve apresentar materiais que enfatizem quer resoluções práticas de problemas (beneficiando os activos) quer a compreensão do fundamento subjacente (beneficiando os reflexivos). Uma técnica geralmente eficaz para envolver os aprendizes activos consiste em pedir aos alunos que, se o desejarem, se organizem em grupos e, periodicamente, sugerir que apresentem respostas colectivas às perguntas propostas pelo professor. Posteriormente as respostas deverão ser partilhadas e discutidas durante o tempo que o professor achar conveniente. Além de forçar a reflexão sobre o material exposto, tais exercícios de *brainstorming* podem indicar aspectos que os estudantes não estejam a compreender e fornecer um ambiente de sala de aula mais agradável do que o que pode ser conseguido com uma apresentação formal. Estas reflexões/discussões normalmente

proporcionam o envolvimento da maioria dos estudantes mais introvertidos, que nunca participariam numa discussão de uma aula segundo um método mais expositivo.

Para alcançar os reflexivos é importante que o professor lance questões de reflexão, deixando sugestões "no ar", sem lhes dar resposta, proporcionando intervalos para a reflexão, incitando assim à imaginação dos reflexivos. Exercícios deste tipo no meio de uma apresentação podem tornar a aula numa experiência educacional estimulante e recompensadora.

Caso um aluno activo se encontre perante um professor que não proporcione tempo de aula para discussões ou actividades de resolução de problemas, o aluno deve tentar compensar estas faltas quando estuda. Assim, deve tentar estudar em grupo, desde que os elementos do grupo se alternem para explicar os diferentes tópicos, uns aos outros.

Quando um aluno reflexivo se encontra perante um professor que não disponibilize tempo para reflexões acerca da nova informação apresentada, o aluno poderá também compensar esta falha durante o seu estudo, por exemplo, não lendo apenas ou memorizando a matéria, mas parando periodicamente para rever o que leu e pensar em possíveis questões ou aplicações. Poderá ser de grande utilidade a realização de resumos das leituras efectuadas ou a elaboração de notas nas aulas, através das suas próprias palavras. O tempo extra dispendido na realização de tais resumos será compensado pela retenção mais eficaz da informação.

Há indicações que sugerem que os engenheiros são naturalmente aprendizes mais activos do que reflexivos. Os activos não aprendem muito em situações que requerem que sejam passivos, como na maioria das apresentações ou aulas teóricas expositivas. Por outro lado, os aprendizes reflexivos não aprendem muito em situações que não forneçam oportunidades de pensar sobre a informação que lhes está a ser apresentada. O facto de um aluno, não ser classificado como activo, não significa que seja passivo. O professor é que pode promover uma aprendizagem activa ou passiva no que concerne à natureza da participação do aluno na aula. Assim, se o professor fomentar uma aprendizagem activa significa que o estudante faz algo na aula, através de um processo de aprendizagem de experimentação activa ou de observação reflexiva. Como se verifica em todas as restantes categorias dos estilos de aprendizagem, quer os aprendizes activos quer os reflexivos são necessários como engenheiros. Os observadores reflexivos são os teóricos, modeladores matemáticos, aqueles que podem definir os problemas e propor possíveis soluções. Os experimentadores activos são os que avaliam as ideias, testam os modelos, projectam e realizam experiências, e encontram soluções que funcionam, são organizadores, responsáveis por decisões. A tabela 3.7 ilustra um quadro resumo com os principais aspectos a considerar face à dimensão Activa/Reflexiva dos estilos de aprendizagem.

Características	
Activos	**Reflexivos**
Gostam de experimentar coisas e fazer algo com a informação no mundo externo.	Gostam de examinar e manipular a informação introspectivamente.
Gostam de trabalhar em equipa.	Gostam de trabalhar a sós.
Experimentalistas.	Teóricos.
Actividades adequadas	
Promover actividades de resolução de problemas.	Promover actividades de reflexão (questões do tipo "se então?" e suas aplicações).
Promover actividades práticas.	Promover actividades teóricas.
Promover actividades de equipa (fóruns de discussão,..).	Promover tarefas individuais.

Tabela 3.7: Caracterização dos Activos/Reflexivos do modelo de Felder-Silverman

3.6.4 Sequencial/Global

Todos os currículos, desde o ensino básico são sequenciais, os livros de texto são sequenciais, e a maioria dos professores ensina de forma sequencial. O desafio de um professor residirá, provavelmente, em alcançar os aprendizes globais. Para tal, o professor deve fornecer o "retrato geral" ou objectivo da lição antes de apresentar as etapas, fazendo o possível para estabelecer o contexto e relevância do assunto ou matéria, relacionando-o com a experiência ou conhecimentos prévios do aluno. Aplicações e questões do tipo "se então" devem ser deliberadamente fornecidas. Porém, é também crucial que o professor dê ao aluno a liberdade de planear o seu próprio método de resolução de problemas não o forçando a adoptar a estratégia que o professor pensa como mais adequada. O professor deve também, frequentemente, expor o aluno a conceitos avançados antes de os apresentar ou explicar. Uma forma particularmente valiosa de os professores "servirem" os aprendizes globais, bem como os sequenciais, consiste em atribuir exercícios de criatividade - problemas conhecidos que envolvam ou possibilitem a geração de soluções alternativas, trazendo assuntos de outros cursos ou disciplinas – e incentivando os estudantes que mostram atitudes promissoras no sentido de os resolver.

Os globais, sentem-se frequentemente desconfortáveis com as aparentes desvantagens do seu método de aprendizagem, e muitas vezes desintegrados. Porém uma ajuda valiosa consiste em lhes explicar o seu próprio processo de aprendizagem. Assim, a revelação desta realidade - a sua criatividade e profundidade de visão – pode ser altamente benéfica, tornando-os mais entusiasmados, menos críticos de si próprios e mais positivos sobre a educação em geral. Se lhes forem dadas oportunidades para mostrarem as suas habilidades

inatas e os seus esforços forem encorajados na escola, poder-se-ão revelar profissionais excepcionais.

Apesar da maioria das disciplinas ser leccionada de forma sequencial, pode acontecer que um aluno sequencial se encontre perante um professor que aprecie saltar de tópico em tópico ou omitir alguns passos de determinada resolução. Neste caso um aluno sequencial deverá ter uma atitude pró-activa, no sentido de perguntar explicitamente ao professor como completar a informação deixada inacabada ou consultando informação que o leve a preenchê-la. Durante o estudo, este tipo de aluno deverá organizar e sublinhar a informação de acordo com um ordenamento lógico e sequencial. No entanto, após este processo, deverá aproveitar para fortalecer as suas capacidades de pensamento global, tentando relacionar cada novo tópico aprendido com os conhecimentos que já possui.

Acontece frequentemente que um professor se esquece de fornecer "o retrato geral" do que vai ensinar, antes das explicações propriamente ditas, o que pode ser desastroso para os alunos globais. Para impedir este tipo de situação, será desejável que um aluno global folheie o capítulo do livro, de forma a ficar com uma perspectiva geral referente à matéria a aprender, antes de começar a estudar e preferencialmente antes de o professor introduzir o assunto. Outra tarefa importante consistirá em dedicar períodos de tempo mais alargados para o estudo e reflexão da matéria de cada disciplina, em vez de um estudo diário distribuído por várias disciplinas. Acima de tudo é importante que estes alunos não percam a esperança e acreditem neles próprios. Apesar de poderem demorar algum tempo a entender determinado assunto, após o conseguirem poderão ver relações com outros assuntos e disciplinas conseguindo aplicá-lo em situações que a maioria dos sequenciais nunca alcançará.

A escola é frequentemente uma experiência difícil para os alunos globais, uma vez que eles não aprendem de uma maneira constante ou previsível e tendem a sentir-se deslocados em relação aos seus colegas e incapazes de ir ao encontro das expectativas dos seus professores. Podem sentir-se "estúpidos" quando se estão a esforçar para dominar os assuntos com os quais a maioria dos seus colegas tem poucas dificuldades. Frequentemente, sentem-se perdidos durante algum tempo, até que alcancem o todo, se "faça luz" e o *puzzle* se resolva. Alguns, eventualmente, desanimam com o ensino e desistem. Contudo, os globais são indivíduos imprescindíveis na sociedade, sintetizadores, investigadores multidisciplinares, pensadores/projectores de sistemas, os que vêem as conexões que mais ninguém vê. Podem ser engenheiros notáveis, se conseguirem sobreviver ao processo educativo vigente. A tabela 3.8 ilustra um quadro resumo com os principais aspectos a considerar face à dimensão Sequencial/Global dos estilos de aprendizagem.

Características	
Sequenciais	**Globais**
Aprendem de forma linear, passo-a-passo.	Aprendem em "saltos" intuitivos.
Seguem caminhos lógicos.	Seguem caminhos aparentemente aleatórios.
Precisam de perceber todos os detalhes, para compreender um problema no seu todo.	Precisam de perceber a relação entre o que lhes está a ser apresentado com os seus conhecimentos prévios, antes de dominar os detalhes.
Pensamento convergente e análise.	Pensamento divergente e síntese.
Actividades adequadas	
Fornecer todas as etapas ou detalhes antes do objectivo geral e contexto.	Fornecer o "retrato geral" ou objectivo da lição antes de apresentar as etapas.
Apresentar o assunto a ensinar de forma lógica e ordenada.	Estabelecer o contexto e relevância do assunto ou matéria, relacionando-o com a experiência prévia do aluno ou situações suas conhecidas.
	Permitir métodos de resolução inovadores.
	Propor exercícios de criatividade.

Tabela 3.8: Caracterização dos Sequenciais/Globais do modelo de Felder-Silverman

3.6.5 Indutivo/Dedutivo

Em geral, os alunos apreciam exposições claras, perfeitamente ordenadas e concisas, especialmente em assuntos relativamente complexos. Também os professores, tradicionalmente, indicam os princípios gerais e vão, gradualmente, detalhando-os no sentido das aplicações, pois este procedimento constitui uma maneira eficiente e elegante de organizar e apresentar a matéria. Porém, esta estratégia resulta para material que já é compreendido, não trazendo geralmente, para a maioria dos alunos, ganhos em termos de aprendizagem de novos assuntos.

Como já mencionado, segundo os autores do modelo de Felder-Silverman, todo o processo de ensino deve ser indutivo. Também variados estudos suportam a ideia de que uma aproximação indutiva do ensino promove uma aprendizagem mais eficaz. Os defensores desta aproximação indicam que esta abordagem permite obter melhores desempenhos académicos e a melhoria das aptidões referentes a raciocínios abstractos (Taba, 1966) maior retenção da informação (McConnell, 1934; Swenson, 1949), melhor capacidade de aplicar princípios (Lahti, 1956), melhor capacidade de resolução de problemas (Kagan, 1965) e aumento das potencialidades para o pensamento inventivo (Chomsky, 1968; Piaget, 1970).

Em geral, uma maneira eficaz para promover uma melhor aprendizagem consiste em seguir o método científico: primeiro indução, depois dedução. Assim, o professor deve preceder as apresentações do material teórico com uma indicação de fenómenos observáveis que a teoria explicará ou de um problema físico que a teoria resolverá. Desta forma infere as regras ou princípios gerais que explicam os fenómenos observados e deduz outras implicações e consequências dos princípios inferidos. Os professores deverão estar cientes que os aprendizes indutivos precisam de motivação para aprender, não sentindo confiança com afirmações do género "Confiem em mim – este assunto ser-vos-á útil um dia" ou "esta teoria tem grandes aplicações práticas que veremos mais à frente". Assim, os problemas propostos deverão consistir primeiramente na apresentação de fenómenos reais e, posteriormente, serem pedidas as regras subjacentes. Tais problemas fortalecem os aprendizes indutivos e ajudam simultaneamente os dedutivos a desenvolver capacidades relativas à sua modalidade de aprendizagem menos preferida, mas mais aconselhada na literatura.

A maioria dos currículos de engenharia são concebidos em torno de linhas dedutivas, começando com fundamentos para estudantes iniciantes e avançando no sentido de projectar algo nos últimos anos. Uma progressão similar é usada normalmente para apresentar a matéria em variados cursos ou módulos: primeiro os princípios e posteriormente (nem sempre) as aplicações. Porém, análises informais sugerem que a maioria dos estudantes da engenharia considera-se aprendizes indutivos, contrapondo os métodos dedutivos habitualmente utilizados pelos professores.

3.7 Conclusões

Muitos professores raramente variam os seus métodos de ensino, não por se acomodarem mas por realmente considerarem que o método por eles adoptado é o melhor face às circunstâncias. Outras vezes tendem a reproduzir um determinado método de ensino que lhes foi ministrado, não questionando sequer a sua adequação. É, no entanto, inegável que o processo de aprendizagem pode condicionar os resultados académicos e de aprendizagem dos alunos. Os métodos de ensino utilizados, o tipo de actividades escolhidas, as características de personalidade do professor poderão afectar a aprendizagem dos alunos. Reconhecer que os alunos são indivíduos diferentes, cada um com as suas particularidades, personalidade própria e os seus próprios estilos de aprendizagem, pode ajudar os professores a adequar os seus métodos de ensino, de forma a proporcionar uma melhor aprendizagem à globalidade dos alunos. É importante que o professor consiga contemplar a enorme diversidade de estilos de aprendizagem, utilizando para isso diferentes estratégias. Adicionalmente, em disciplinas particularmente exigentes como é o caso das programações é

igualmente desejável que o professor se assegure que os alunos adoptam a estratégia mais apropriada para cada assunto.

Numerosos modelos de estilos de aprendizagem têm sido desenvolvidos ao longo dos anos. Existe alguma sobreposição entre alguns deles e certos modelos derivam de outros. Diferentes teorias e argumentos estão na base da escolha de cada modelo. Acreditamos, no entanto, que o mais importante é o conhecimento da existência desta temática e a colocação em prática de estratégias consonantes. Um professor que conheça os seus próprios estilos e possa detectar determinados aspectos dos estilos dos seus alunos certamente melhor será capaz de conceber cursos mais adequados e de melhor trabalhar com os seus alunos. Este conhecimento, por parte dos professores, poderá adicionalmente ajudar o professor a, numa fase mais precoce, identificar e resolver problemas de aprendizagem dos seus alunos, intervindo no sentido de os ajudar a tornarem-se aprendizes mais eficazes.

Também para os alunos consideramos importantíssimo o conhecimento, compreensão e divulgação desta temática. A consciência, por parte dos alunos, de que os seus desempenhos poderão ser devidos a determinadas características pessoais como os seus estilos de aprendizagem pode ajudá-los a uma melhor compreensão e aceitação pessoal. A compreensão dos prováveis pontos fracos e possíveis tendências ou hábitos que podem estar na origem de determinadas dificuldades académicas, poderá fornecer indicações valiosas de estudo e comportamentos para as ultrapassar.

De referir que não é nossa intenção rotular os alunos. Também não há evidências documentadas de que um conjunto de preferências ou estilos de aprendizagem sejam academicamente superiores a outros. Um determinado perfil de estilo de aprendizagem não reflecte necessariamente a adequação ou inadequação do aluno relativamente a determinada matéria, curso ou profissão. O perfil do estilo de aprendizagem de um aluno fornece apenas uma indicação dos prováveis pontos fortes e possíveis tendências ou hábitos que podem estar a conduzir a determinadas dificuldades na vida académica. Todos os tipos e estilos são únicos. Porém, a consciência deste facto poderá fornecer indicações preciosas para um ensino e aprendizagem mais eficazes.

Também, não se está a sugerir que determinada instituição deva adaptar o seu ensino para "servir" os estilos de aprendizagem de cada aluno, mas apenas a alertar para uma consciência relativa à diversidade de estilos de aprendizagem, de forma a proporcionar a todos os estudantes igualdade de oportunidades. De forma idêntica, não estamos convictos de que um indivíduo deva apenas ser sujeito às condições que lhe são favoráveis. Acreditamos até que talvez a aprendizagem seja mais eficaz, quando um indivíduo usa as capacidades de todos os tipos de aprendizes e seja sujeito a determinadas adversidades. Também sabemos

que, em muitas situações, todos utilizamos ambas as faculdades relativas a cada dimensão de determinado modelo de estilos de aprendizagem. Porém, a maioria das pessoas tende a obter melhores desempenhos se as suas categorias preferidas forem potenciadas ou pelo menos sentir-se-ão mais motivadas nas situações que as favorecem, pelo que estes aspectos deverão ser considerados.

Cap. 4

Taxonomias

"A primeira condição para ser alguma coisa é não querer ser tudo ao mesmo tempo."

Tristão de Ataíde

4.1 Introdução

São diversos os factores que poderão contribuir para o sucesso da aprendizagem de determinado assunto. O ensino e a aprendizagem de assuntos complexos, como a programação, requerem um conjunto de cuidados adicionais. As características do ensino tradicional e actual tornam complicada a existência de um apoio personalizado ao aluno. O ensino é demasiado homogéneo não existindo tempo nem condições para perceber as dificuldades individuais de cada aluno. Por outro lado, o ensino de programação consiste basicamente em duas etapas, a primeira carregada de detalhes sintácticos onde é suposto que os alunos absorvam uma grande quantidade de regras sem que tenham tempo para perceber a sua aplicação. A segunda, onde é esperado que os alunos comecem por resolver problemas de programação, construindo programas completos. Este aspecto implica que o aluno esteja munido, desde uma fase inicial, de um conjunto de capacidades práticas, sem que tenha tempo para as desenvolver. Relativamente a este assunto, refira-se David Gries (1974) que estabelece a seguinte analogia:

"Suponha que frequenta um curso de carpintaria. O instrutor mostra-lhe momentaneamente uma serra, uma plaina, um martelo e algumas outras ferramentas, deixando-o experimentá-las por alguns minutos. Em seguida mostra-lhe um bonito armário terminado. Finalmente, diz-lhe para projectar e construir o seu próprio armário. Certamente pensaria que ele era louco!"

De acordo com este autor, nem a feitura de armários nem a realização de programas, podem ser ensinadas através da simples explicação do funcionamento das ferramentas e consequente demonstração de produtos acabados. Este autor refere ainda que tradicionalmente, no início de uma disciplina introdutória de programação, é apresentado ao aluno um problema seguido por uma apresentação de um programa para o resolver e, posteriormente, são analisados os elementos da solução apresentada. Do ponto de vista de um aluno, o programa foi desenvolvido numa única etapa, começando por uma especificação do problema e tendo por resultado uma solução de funcionamento. De acordo com este autor, o facto de todos nós começarmos por desenvolver soluções parciais e menos eficientes que posteriormente refinamos e melhoramos, parece ser um dos segredos mais bem guardados do ensino da programação. Cremos ser de extrema importância, que os professores mostrem aos alunos a forma de resolver problemas de programação, bem como as etapas necessárias, chamando a atenção para as estratégias utilizadas e a forma de ultrapassar as dificuldades, que com persistência, esforço e prática vão sendo dissipadas.

Pensamos que o ensino da programação deva ser feito através da proposta de exercícios, onde gradualmente sejam colocados aos alunos desafios de intensidade crescente, de acordo com o seu nível cognitivo. Por outro lado, também não defendemos que os exercícios propostos sejam demasiado fáceis, o que geralmente se traduz em desmotivação dos alunos. Para consolidarmos esta opinião, consultámos a literatura sobre sistemas que pudessem suportar uma aprendizagem adequada aos alunos e encontrámos resposta nas taxonomias educacionais. Pensamos que a utilização deste tipo de instrumentos acrescentará um conjunto de valores importantíssimo aos métodos de ensino tradicionais. Assim, a possibilidade de realização de actividades de ensino/aprendizagem mais adequadas ao nível cognitivo do aluno, poderá aumentar a sua motivação. O desinteresse manifestado pelos alunos nas salas de aula poderá também estar relacionado com a sua total incapacidade de acompanhamento dos exercícios que estão a ser abordados, por estes serem cognitivamente desajustados. A utilização de uma taxonomia possibilitará também, ao próprio aluno adquirir consciência sobre os aspectos que ainda não estão bem dominados e assim, seguir um estudo mais direccionado para as suas dúvidas específicas. Simultaneamente, o aluno poderá interiorizar o sentimento de confiança ao ir ultrapassando determinadas etapas com sucesso.

Uma taxonomia é um sistema de classificação ordenado de acordo com determinados critérios. A taxonomia de Linnaeus foi a primeira taxonomia de referência, classificando os organismos vivos numa hierarquia estruturada em árvore. Este ordenamento proporcionou aos biólogos uma ferramenta para os ajudar a compreender o relacionamento entre membros dos reinos animal e vegetal e a expressarem-se com exactidão sobre eles (Biggs, 1999).

As taxonomias de objectivos educacionais podem igualmente ser usadas para fornecer uma linguagem comum para descrever resultados de aprendizagem e desempenhos nas avaliações, reflectindo o estágio de aprendizagem de um aluno. Em geral, as taxonomias objectivam especificar um caminho pelo qual o aprendiz tem de passar de forma a alcançar determinado nível de conhecimento. Um nível mais elevado de conhecimento deve ser construído sobre o de nível inferior. Isto significa que o aprendiz que atinja determinado nível de conhecimento deve demonstrar proficiência em qualquer dos níveis inferiores da taxonomia.

A generalidade das taxonomias divide os objectivos educacionais em três domínios: cognitivo, afectivo e psicomotor. O domínio cognitivo está relacionado com capacidades de aprendizagem mental. O domínio afectivo está relacionado com a aprendizagem de sentimentos ou aspectos da área emocional, expressos como interesses, atitudes ou valores. O domínio psicomotor está relacionado com capacidades físicas ou manuais. Trata de objectivos que enfatizam alguma habilidade muscular ou motora.

Como descrito em Fuller et al. (2007), algumas taxonomias, de que é exemplo a taxonomia de Bloom (Bloom et al., 1956), tratam cada um dos domínios como uma série contínua de uma única dimensão, enquanto outras, como a taxonomia de Bloom revista (Anderson et al., 2001), descrevem o domínio cognitivo usando uma matriz. Contudo outras, como a taxonomia SOLO (Biggs e Collis, 1982), usam um conjunto de categorias que descrevem uma mistura de diferenças quantitativas e qualitativas sobre os desempenhos do aluno. Há também taxonomias que usam igualmente todos os três domínios. Porém, a pesquisa existente sobre a utilização de taxonomias de aprendizagem em ciências da computação centra-se no domínio cognitivo.

De seguida serão apresentadas diversas taxonomias existentes, os seus propósitos, bem como a sua utilização e aplicação em geral e ao ensino da programação em particular. As taxonomias apresentadas são as consideradas principais no estudo conduzido pelo grupo de trabalho de Fuller e colegas (Fuller et al., 2007).

4.2 Taxonomias do Domínio cognitivo

O domínio cognitivo é o âmbito do saber. Fazem parte do domínio cognitivo, operações mentais como a descoberta ou reconhecimento de informação, a retenção ou armazenamento de informação, a geração de informações a partir de certos dados e a tomada de decisão ou feitura de julgamento acerca da informação (Magill, 1980). O domínio cognitivo inclui objectivos vinculados à memória ou recognição e ao desenvolvimento de

capacidades e habilidades intelectuais. Inclui objectivos que enfatizam relembrar ou reproduzir algo que foi aprendido ou que envolvem a resolução de alguma actividade intelectual para a qual o indivíduo tem que determinar o problema essencial, após o que conseguirá reorganizar o material ou combinar ideias, métodos ou procedimentos previamente aprendidos.

4.2.1 Taxonomia de Bloom e Taxonomia de Bloom revista

De entre as diversas taxonomias propostas na literatura, a taxonomia original de Bloom surge como uma referência incontornável, por ter sido a primeira taxonomia descrita como um modelo hierárquico para o domínio cognitivo (Bloom et al., 1956). É uma classificação de diferentes níveis de objectivos e competências que os educadores definem para que os alunos alcancem determinado nível cognitivo. Nesta taxonomia, as competências referentes ao domínio cognitivo são divididas em seis níveis ou categorias, de exigência cognitiva crescente, nomeadamente, Conhecimento, Compreensão, Aplicação, Análise, Síntese e Avaliação (tabela 4.1).

Nível Cognitivo
1. Conhecimento
2. Compreensão
3. Aplicação
4. Análise
5. Síntese
6. Avaliação

Tabela 4.1: Níveis cognitivos da Taxonomia de Bloom original

Considera-se que em cada nível ou categoria os processos são cumulativos, ou seja, uma categoria cognitiva depende da anterior e, por sua vez, dará suporte à seguinte. A edificação das categorias sobre as de níveis inferiores supõe que os níveis mais elevados são considerados mais complexos e próximos do conhecimento absoluto de um determinado conteúdo ou matéria. Os processos caracterizados pela taxonomia devem representar resultados de aprendizagem, ou seja, cada categoria representa o que o indivíduo aprende, não aquilo que ele já sabe. As categorias são também organizadas num gradiente em termos de complexidade dos processos mentais.

O Conhecimento consiste em reconhecer ou recordar informação, princípios específicos e universais, métodos, processos, testes padrão, estruturas ou conjuntos. A lista de palavras que Bloom definiu para o tipo de actividades neste nível inclui "memorizar", "nomear" ou "reconhecer". Reportando-nos ao caso concreto de programação introdutória, Lister e Leaney (2003b) referem que perguntas de programação sobre Java, adequadas ao nível do conhecimento, seriam por exemplo: "Indique os três tipos de ciclos existentes em Java." ou "Que tipo de ciclos em Java é que são executados pelo menos uma vez?"

A Compreensão consiste em demonstrar entendimento suficiente para organizar e arranjar mentalmente a informação sobre determinado assunto. Neste nível, o aluno compreende o que está a ser comunicado sem que necessariamente o relacione com outros assuntos ou veja as suas implicações na plenitude. A lista de palavras que Bloom definiu para o tipo de actividades neste nível inclui "explicar" ou " traduzir". Relativamente à programação inicial, Lister e Leaney (2003b) referem que uma pergunta de programação sobre Java, adequada ao nível de compreensão poderia consistir na tradução para Java de uma instrução muito específica descrita em formato textual ou pseudocódigo. Por exemplo, "Escreva uma instrução em Java que declare uma variável x que possa conter valores inteiros".

A Aplicação consiste em alcançar uma resposta, através da aplicação da informação previamente aprendida. Bloom refere que o nível de aplicação consiste na utilização de abstracções em particular e situações concretas, podendo incluir ideias gerais, regras de procedimentos, métodos generalizados, princípios técnicos, ideias e teorias que devem ser recordadas e aplicadas. A lista de palavras que Bloom definiu para o tipo de actividades neste nível inclui "calcular", "resolver" ou "escrever". Lister e Leaney (2003b) referem que muitas tarefas tradicionais de programação são tarefas do nível de aplicação, desde que a tarefa seja completamente especificada. Reportando-se ao caso concreto de programação introdutória, estes autores referem que uma pergunta de programação sobre Java, adequada ao nível de aplicação poderia consistir em pedir aos alunos para escrever uma classe, desde que as características da classe fossem especificadas no enunciado de uma forma muito detalhada relativamente aos cabeçalhos dos métodos e às variáveis membro e desde que o código de cada método não fosse longo (por exemplo, menor que 20 linhas) ou algoritmicamente complexo.

A Análise consiste em perguntas exigindo pensamento crítico e detalhado. Bloom refere que o nível de análise enfatiza o particionamento do assunto nas suas partes constituintes, bem como a detecção dos relacionamentos entre elas e a forma como são organizadas. A lista de palavras que Bloom definiu para o tipo de actividades neste nível inclui "categorizar", "diferenciar", "discriminar" ou "distinguir". Lister e Leaney (2003b)

referem que uma tarefa apropriada de avaliação sobre Java adequada ao nível de análise poderia, por exemplo, consistir em fornecer ao aluno código que implementa o padrão de programação Modelo-Controlo-Vista, e pedir-lhe que identifique as partes do código que implementam o modelo, a vista e o controlador.

A Síntese inclui perguntas exigindo a execução de pensamento original e criativo. Consiste em juntar todas as partes para formar o todo. A lista de palavras que Bloom definiu para o tipo de actividades neste nível inclui "criar", “projectar”, “organizar” ou “planear”. Lister e Leaney (2003b) referem que, tal como no nível de aplicação, muitas perguntas típicas de programação se situam neste nível. A diferença de uma tarefa de aplicação relativamente a uma de síntese é que esta contém escolhas de projecto mais sofisticadas. Em geral as perguntas deste nível consistem em descrições vagas de problemas, que os alunos têm de refinar e implementar. Para o caso de programação em Java estes autores referem que um exercício típico deste nível poderia consistir em decompor um problema em classes e determinar os métodos mais apropriados a cada classe. Bloom definiu o nível de síntese como o “habilidade para a escrita, usando uma excelente organização das ideias”. Apesar de se referir à escrita de relatórios, Lister e Leaney (2003a) referem que esta definição se encaixa perfeitamente na programação, na medida em que a programação exige uma certa criatividade para a composição dos programas.

A Avaliação inclui perguntas abertas que possibilitem várias respostas ou soluções. Bloom definiu o nível de avaliação como consistindo em fazer julgamentos acerca do valor de determinada solução, ideia ou trabalho, em termos de exactidão, eficácia ou satisfação. De acordo com Lister e Leaney (2003a) este nível da taxonomia adequa-se perfeitamente ao ensino da programação, na medida em que neste tipo de disciplina se valorizam aspectos como código claro, conciso e eficiente. A lista de palavras que Bloom definiu para o tipo de actividades neste nível inclui "avaliar" ou “criar”. Os mesmos autores referem que este nível da taxonomia se mapeia facilmente no ensino da programação. Os autores referem que uma tarefa de programação sobre Java adequada ao nível de avaliação poderia, por exemplo, consistir em os alunos reverem determinado código Java fornecido, avaliando a sua eficácia.

Os níveis da hierarquia, bem como exemplos de verbos utilizados para definir actividades adequadas à aquisição de capacidades em cada nível, encontram-se resumidos na figura 4.1 e na tabela 4.2.

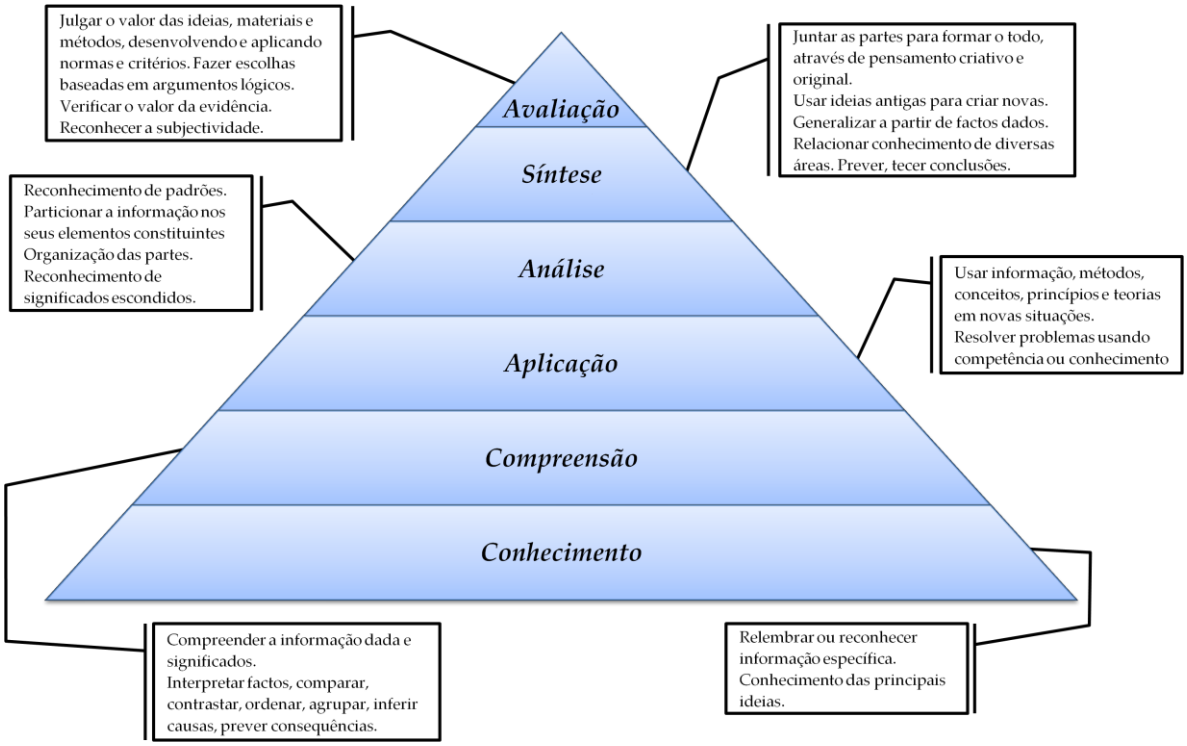


Figura 4.1: Níveis hierárquicos da Taxonomia de Bloom e correspondentes actividades

A taxonomia de Bloom foi revista por Anderson e um grupo de investigadores da psicologia cognitiva (Anderson et al., 2001). As duas versões da taxonomia de objectivos educacionais são frequentemente descritas na literatura como Taxonomia de Bloom e Taxonomia de Bloom revista. Como resultado, foram feitas alterações em termos de terminologia e estrutura. As alterações de terminologia fizeram com que os nomes listados no modelo de Bloom original fossem substituídos por verbos, de forma a corresponderem às formas pelas quais os objectivos de aprendizagem são vulgarmente descritos. Esta informação encontra-se ilustrada na tabela 4.3.

Exemplos de verbos que representam actividades intelectuais em cada nível[3]

Conhecimento	Compreensão	Aplicação	Análise	Síntese	Avaliação
Declarar	Comparar	Aplicar	Analisar	Construir	Avaliar
Definir	Demonstrar	Calcular	Atribuir	Criar	Criticar
Identificar	Descrever	Classificar	Contrastar	Discutir	Defender
Listar	Explicar	Manipular	Deduzir	Escrever	Escolher
Mostrar	Interpretar	Modificar	Diferenciar	Esquematizar	Estimar
Nomear	Parafrasear	Relacionar	Distinguir	Formular hipóteses	Julgar
Reconhecer	Reafirmar	Resolver	Escolher	Planear	Justificar
Relembrar	Sumariar	Usar	Organizar	Projectar	Verificar

Tabela 4.2: Lista de verbos para os diferentes níveis hierárquicos da Taxonomia de Bloom

Categorias	Processos Cognitivos
1. Recordar	Reconhecer, Recordar
2. Compreender	Interpretar, Exemplificar, Sumariar
3. Aplicar	Executar, Implementar
4. Analisar	Diferenciar, Organizar, Atribuir
5. Avaliar	Verificar, Criticar
6. Criar	Gerar, Planear, Produzir

Tabela 4.3: Processos cognitivos da Taxonomia de Bloom revista

Recordar, consiste em reconhecer e recordar informações importantes presentes na memória de longa duração. Compreender é a capacidade de fazer a sua própria interpretação do material educacional, por exemplo acerca de leituras ou de explicações do professor. Dentro do processo de compreensão é possível incluir capacidades de interpretação, exemplificação, classificação, resumo, conclusão, comparação e explicação. Aplicar, refere-se à utilização de determinado procedimento já aprendido a uma situação familiar ou nova. Analisar, consiste em dividir o conhecimento em partes e pensar como essas partes se relacionam com a estrutura geral. A análise dos alunos é feita por meio de diferenciação, organização e atribuição. Avaliar, o nível mais avançado da taxonomia original, é o quinto nesta versão revista e engloba, na sua essência, as capacidades de verificação e crítica. Criar, um processo que não fazia parte da taxonomia original, é a componente mais complexa da nova versão. Esta nova capacidade envolve reunir elementos para dar origem a algo novo. Para conseguir criar tarefas, os alunos têm de gerir, planear e produzir.

[3] Fonte: Tradução de *Table of verbs for Bloom's Taxonomy.* Acedido em Janeiro de 2010, de Bendigo Senior Secondary College's web site: http://edtech.clas.pdx.edu/presentations/frr99/blooms.htm

Estas taxonomias não definem uma sequência de ensino, mas antes os níveis de desempenho que é esperado que o aluno obtenha face a determinado conteúdo. É esperado que um aluno que consiga desempenhar as tarefas de um nível mais elevado também possa executar as dos níveis inferiores da hierarquia. Este aspecto poderia ser interpretado como a implicação de um processo de aprendizagem sequencial. Porém, a taxonomia não prevê a utilização de uma aproximação iterativa da aprendizagem de determinado conteúdo. Os autores da taxonomia revista reconhecem até que poderá haver sobreposição em termos da complexidade cognitiva entre as categorias de mais alto nível da hierarquia. Contudo, muitos autores consideram que o ponto médio de cada uma das categorias de nível mais elevado deva exigir capacidades cognitivas mais complexas do que as requeridas pela categoria mais baixa (Krathwohl et al., 1964; Svec, 2005). Por exemplo, em alguns contextos, o processo cognitivo de exemplificação na categoria Compreender pode exigir uma carga cognitiva mais elevada do que a execução na categoria Aplicar, mas em geral isso não deve acontecer.

As alterações da estrutura consideram uma gama maior de factores que afectam o ensino e a aprendizagem, tentando corrigir alguns problemas da taxonomia original. Em particular, esta taxonomia diferencia o "saber o quê" (o conteúdo) do "saber como" (os procedimentos para resolver problemas). A dimensão do conhecimento divide-se em quatro categorias: Factual, Conceptual, Procedimental e Metacognitiva. O conhecimento factual inclui elementos isolados de informação, como definições de vocabulário, conhecimento de terminologia e conhecimento de detalhes e elementos específicos. O conhecimento conceptual consiste no conhecimento de classificações e categorias, princípios e generalizações, teorias, modelos e estruturas. O conhecimento procedimental (saber como fazer) inclui o conhecimento de capacidades específicas de determinado assunto, algoritmos, heurísticas ou métodos empíricos, técnicas e métodos específicos sobre determinado assunto, bem como o conhecimento sobre os critérios para determinar quando usar esses procedimentos. O conhecimento metacognitivo (reflectir sobre o que se sabe) refere-se ao conhecimento dos processos cognitivos e das informações sobre como manipular esses processos de forma eficaz. Implica um conhecimento estratégico e sobre as tarefas cognitivas, incluindo o conhecimento contextual e condicional apropriado. Implica também um autoconhecimento, de forma a utilizar a estratégia ou ferramenta mais adequada aos estilos de entendimento do indivíduo. Esta divisão do conhecimento fornece uma matriz onde os objectivos de aprendizagem são mapeados, como ilustrado na tabela 4.4.

	A. Factual	B. Conceptual	C. Procedimental	D. Metacognitivo
1. Recordar				
2. Compreender				
3. Aplicar				
4. Analisar				
5. Avaliar				
6. Criar				

(adaptada de Fuller et al., 2007)

Tabela 4.4: Processos cognitivos e dimensão do conhecimento da Taxonomia de Bloom revista

De acordo com esta taxonomia, cada nível de conhecimento pode corresponder a um nível do processo cognitivo, portanto o aluno pode recordar um conhecimento factual ou procedimental, compreender um conhecimento conceitual ou metacognitivo ou analisar um conhecimento metacognitivo ou factual.

4.2.2 Taxonomias de Niemierko e Tollingerova

Apesar das taxonomias mais conhecidas serem a taxonomia de Bloom e a Taxonomia de Bloom revista, outras taxonomias de objectivos educacionais constituíram uma extensão das anteriores. Na Eslováquia, por exemplo, a taxonomia de Niemierko é muitas vezes citada, sendo por vezes apresentada como mais adequada para a construção de testes cognitivos (para avaliação de conhecimento). Niemierko refere que, em disciplinas da área de ciências, as três categorias mais elevadas da Taxonomia de Bloom (processos de pensamento mais elevados) não podem ser ordenadas hierarquicamente. Face a estas convicções, Niemierko desenvolveu a Taxonomia "ABC" dos objectivos educacionais organizada em quatro categorias, agrupadas em dois níveis principais ou dimensões, o nível de Conhecimento e o nível de Capacidades e Competências, como se mostra na tabela 4.5.

Níveis	**Categorias dos objectivos de aprendizagem**
I. Conhecimento	A. Recordar o conhecimento B. Compreender o conhecimento
II. Capacidades e Competências	C. Aplicar o conhecimento a problemas típicos D. Aplicar o conhecimento a problemas desconhecidos

Tabela 4.5: Níveis da Taxonomia "ABC" dos objectivos educacionais de Niemierko

No nível de Conhecimento encontram-se a categoria de Recordar (o conhecimento), similar à categoria de Conhecimento da taxonomia de Bloom e Compreender (o

conhecimento), similar à categoria de Compreensão da taxonomia de Bloom. No nível de Capacidades e Competências encontram-se duas categorias referentes a Aplicar o conhecimento a problemas típicos e a Aplicar o conhecimento a problemas desconhecidos. Por vezes estas categorias surgem designadas como Transferência específica (aplicação da informação adquirida de acordo com padrões apresentados) e Transferência não específica (aplicação criativa da informação adquirida). A categoria de Transferência específica corresponde à categoria de Aplicação da taxonomia de Bloom. A categoria de Transferência não específica inclui as categorias: Análise, Síntese e Avaliação da taxonomia de Bloom.

Também Tollingerova classificou as tarefas de aprendizagem em cinco categorias ordenadas hierarquicamente, tendo como base a taxonomia de Bloom e atendendo às exigências cognitivas das tarefas de aprendizagem (Kundratova e Turek, 2001). As categorias são:

1. Reprodução da memória do conhecimento. Tarefas de aprendizagem que exigem a reprodução do conhecimento, utilizada quando os alunos empregam operações de memória.

2. Operações fáceis do pensamento com conhecimento. Tarefas de aprendizagem que exigem operações mentais simples de conhecimento, tal como observar, resumir, comparar e categorizar.

3. Operações difíceis do pensamento com conhecimento. Tarefas de aprendizagem exigindo operações mentais complexas de conhecimento, tal como indução, dedução, interpretação, transformação ou verificação.

4. Comunicação do conhecimento. Tarefas de aprendizagem exigindo conhecimento de interpretação, utilizado quando os alunos interpretam não apenas os resultados da sua própria solução mas também o seu progresso, condições e fases.

5. Pensamento criativo. Tarefas de aprendizagem que exigem pensamento criativo baseado em operações prévias, bem como a capacidade de combinar essas operações em complexos mais alargados, resultando em novas soluções.

4.3 Taxonomias de Pensamento Crítico

Alguns investigadores consideram que a taxonomia de Bloom não dá ênfase suficiente aos aspectos do pensamento crítico. Segundo eles, o pensamento crítico vai além das categorias cognitivas da taxonomia original de Bloom, incorporando atributos de julgamento reflexivo no que diz respeito ao valor do que está a ser ensinado e permitindo fazer julgamentos sobre a fiabilidade e autoridade do conhecimento associado. O pensamento crítico consiste na capacidade de pensar clara e racionalmente. Inclui a capacidade de um indivíduo se envolver em pensamento reflexivo e independente. Deste tipo de taxonomias cite-se por exemplo, a taxonomia do julgamento reflexivo de King e Kitchener (1994) e a taxonomia do pensamento crítico de Facione (1984).

A taxonomia do julgamento reflexivo de King e Kitchener apresenta um total de sete estágios que se enquadram em três grupos indicativos do pensamento pré-reflexivo (estágios 1-3), pensamento quasi-reflexivo (estágios 4 e 5) e pensamento reflexivo (estágios 6 e 7). De acordo com os seus autores, esta taxonomia permite capturar o desenvolvimento das capacidades de raciocínio dos alunos. O processo de avaliação é baseado na apresentação de situações complexas aos alunos, analisando o seu pensamento sobre problemas mal definidos. É claro que este método exige tempo intensivo quer da parte do aluno quer da parte do analisador.

Cada estádio representa um conjunto de posicionamentos que o indivíduo assume, com um certa coerência lógica interna, organizados numa sequência hierárquica de complexidade crescente. Cada conjunto de convicções reflecte não apenas a forma como os indivíduos percebem a natureza do conhecimento e do modo como ele pode ser obtido, mas também o tipo de estratégias que utilizam para resolver problemas mal definidos. À medida que os indivíduos avançam no seu percurso desenvolvimental, tornam-se mais capazes de avaliar a natureza do próprio conhecimento, e de explicar e defender os seus próprios pontos de vista em assuntos controversos.

No primeiro estádio o indivíduo percebe o conhecimento como absoluto e certo, sendo limitado a experiências e observações simples e concretas: é o que a pessoa observa que é verdadeiro. No segundo estádio já se considera que a certeza absoluta pode sofrer algumas alterações, de modo a lidar com alguma incerteza que decorre da inacessibilidade de certo tipo de conhecimento. No terceiro estádio emerge a possibilidade de nalgumas áreas o conhecimento poder ser temporariamente incerto, pelo que se deve recorrer ao conhecimento das crenças pessoais. O quarto estádio é dominado pela incerteza. De acordo com as autoras desta taxonomia, uma das maiores transformações no pensamento ocorre neste estádio à medida que a crença na incerteza do conhecimento se estrutura. É no quinto estádio que a

sustentação do conhecimento deixa de ser pessoal, passando a derivar das regras de disciplinas específicas (áreas do conhecimento), ou seja, é neste estádio que são interpretadas evidências decorrentes de regras de contextos específicos. O verdadeiro julgamento reflexivo só é possível nos estádios seis e sete na medida em que o conhecimento não é fornecido mas tem de ser construído. Apesar de no estádio seis os indivíduos avaliarem diferentes fontes para construírem soluções para os problemas, acreditam que estas soluções são individuais e limitadas pelas suas próprias visões da situação específica onde o problema surgiu. O sétimo estádio diferencia-se do anterior porque os indivíduos acreditam que o conhecimento é um processo contínuo e continuado de questionamento/inquérito e reavaliação. Neste estádio considera-se que a melhor solução para os problemas é circunscrita, contextualizada e provisória de modo a que outras pessoas sensatas e racionais que considerem a evidência a possam compreender. Apesar de o julgamento reflexivo efectivo só ser possível nos últimos estádios, cada estádio possui modos cada vez mais inter-relacionados, complexos e efectivos de resolver problemas mal definidos. No seu livro (King e Kitchener, 1994), as autoras discutem aspectos do pensamento, os desafios com que os alunos se defrontam e sugerem actividades de aprendizagem para desafiar os alunos em cada estádio.

A taxonomia do pensamento crítico de Facione encontra-se mais próxima da taxonomia de Bloom. Facione considera que o pensamento crítico consiste no desenvolvimento e avaliação de argumentos. Facione et al. (1994) apresentam uma definição de consenso sobre o pensamento crítico, resultado de um projecto de pesquisa encetado por investigadores especializados e teóricos multidisciplinares. Concluíram que pensamento crítico é o julgamento intencional, auto-regulador que resulta em interpretação, análise, avaliação e inferência, assim como a explicação de evidências conceituais, metodológicas, criteriológicas ou contextuais nas quais o julgamento foi baseado. De acordo com estes autores, para se transformar num bom pensador crítico que exiba auto-regulação, o indivíduo deve ser proficiente na interpretação, análise, avaliação, inferência, explanação e na auto-regulação metacognitiva. Para Facione o pensamento crítico é a habilidade de pensar de forma clara e racional. Inclui a capacidade de um indivíduo se envolver no pensamento reflexivo e independente. Facione lista seis capacidades que um indivíduo com pensamento crítico deve exibir, nomeadamente: compreender as conexões lógicas entre ideias; identificar, construir e avaliar argumentos; detectar inconsistências e erros comuns de raciocínio; resolver problemas sistematicamente; identificar a relevância e a importância das ideias e reflectir sobre a justificação relativa às opiniões pessoais, crenças e valores.

4.4 Taxonomias de Domínio Unificado

Tem havido várias tentativas no sentido de produzir uma taxonomia que cubra os domínios Cognitivo (C), Afectivo (A) e Psicomotor (P). O trabalho de De Block (Kundratov e Turek, 2001) é um exemplo desta abordagem. A taxonomia resultante é constituída por quatro níveis, nomeadamente Conhecimento, Compreensão, Aplicação e Integração, definindo para cada um deles as capacidades referentes a cada um dos domínios Cognitivo, Afectivo e Psicomotor. Esta informação pode ser consultada, de forma resumida na tabela 4.6.

Nível	Características
Conhecimento	**C**: Repetir, definir, mostrar, nomear. **A**: Escutar a opinião dos outros, registar notas, fazer constatações. **P**: Mostrar, imitar, compreender o som, o cheiro ou o gosto.
Compreensão	**C**: Descrever, caracterizar, dizer através de palavras próprias, explicar, comparar. **A**: Aceitar opiniões de outros, responder a perguntas, reagir a regras correctamente, fazer perguntas relevantes, participar. **P**: Demonstrar um princípio, compor ou decompor algo que é conhecido.
Aplicação	**C**: Resolver, calcular, numerar, traduzir, ilustrar, analisar, fazer. **A**: Reagir às regras automaticamente, aceitar normas e valores, cooperar num grupo, aplicar normas e regras. **P**: Fazer, produzir, experimentar, reparar, adaptar, juntar ou desmontar/decompor algo que é novo.
Integração	**C**: Projectar, criar, sumariar, julgar, decidir, planear. **A**: Reagir às regras espontaneamente, aplicar normas espontaneamente e comportar-se sob determinadas regras, cooperar, encontrar satisfação no comportamento e no trabalho sob regras da sociedade. **P**: Executar uma actividade fluentemente, sem hesitação, sem erros, automaticamente; trabalhar de forma precisa, rapidamente.

Tabela 4.6: Níveis da Taxonomia de Domínio Unificado

4.5 Taxonomia SOLO

A taxonomia SOLO (*Structure of the Observed Learning Outcome*) não faz referência às características cognitivas de desempenho do aluno ou à dimensão afectiva (Biggs e Collis, 1982). Centra-se no conteúdo da resposta do aluno relativamente ao que está a ser avaliado. Consiste numa tentativa para identificar a natureza de determinado conteúdo e dos relacionamentos estruturais dentro desse conteúdo. O conteúdo pode ser planeado para avaliar o conhecimento, capacidades cognitivas ou valores subjacentes. A taxonomia pode ser usada para estabelecer os relacionamentos esperados entre os diferentes tipos de conteúdo. É deixado a quem planeia o curso o tipo de conteúdo que se pretende avaliar. Os níveis da taxonomia SOLO são: Pre-estrutural, Uni-estrutural, Multi-estrutural, Relacional e Abstracto.

O nível Pre-estrutural consiste no tipo menos sofisticado de resposta que um aluno pode dar. Revela insuficiente conhecimento sobre um assunto ou um conhecimento pouco relacionado ou até desgarrado, insuficiente para realizar uma tarefa adequadamente. Relativamente à interpretação de um pequeno pedaço de código de programação, Lister et al. (2006) referem que um estudante que dê uma resposta pré-estrutural manifesta um desentendimento (*misconception*) significativo do tópico de programação em questão ou então utiliza um conceito prévio irrelevante para o assunto.

O nível Uni-estrutural revela um tipo de resposta em que o aluno manifesta uma certa compreensão sobre alguns mas não todos os aspectos de determinado problema. O aluno percebe determinados significados simples, revelando uma compreensão parcial de determinado problema dentro de um caso mais complexo. Normalmente a resposta do aluno foca-se apenas num aspecto relevante da tarefa a resolver.

O nível Multi-estrutural revela um tipo de resposta em que o aluno manifesta uma compreensão de várias ou todas as partes do problema, mas não uma consciência dos relacionamentos entre elas, tratando-as independentemente. Lister et al. (2006) referem que este tipo de conhecimento é tipo uma "lista de compras", ou seja, uma colecção desorganizada de elementos. Estes autores utilizam ainda a seguinte metáfora, os alunos deste nível conseguem ver as árvores mas não a floresta. Referem ainda que relativamente à interpretação de determinado código os alunos poderão, por exemplo, perceber qual o valor final com que fica determinada variável, mas não compreender o que faz o código, onde essa variável está inserida, na sua totalidade. De acordo com estes autores, uma resposta deste nível poderá estar correcta ou incorrecta, mas a técnica usada ser multi-estrutural.

O nível Relacional revela um tipo de resposta em que o aluno apresenta um nível de compreensão capaz de integrar as partes do problema numa estrutura coerente, usando essa estrutura para resolver determinada tarefa. O aluno é capaz de usar conceitos que integram determinado conjunto de dados compreendendo a sua aplicação a problemas familiares. De acordo com Lister et al. (2006), um aluno que forneça este tipo de respostas já consegue ver a floresta, para além das árvores. Estes autores referem que relativamente à interpretação de determinado código os alunos poderão, por exemplo, perceber o seu comportamento geral, sem que tenham de fazer a sua simulação manual. Tal como no nível Multi-estrutural, uma resposta deste nível poderá igualmente estar correcta ou incorrecta, mas a abordagem usada ser relacional.

O nível Abstracto revela um tipo de resposta em que o aluno apresenta um nível de compreensão que possibilita o relacionamento entre os princípios existentes e conhecidos, de modo a que possa lidar com problemas novos (Biggs e Collis, 1982; Biggs, 1999). Os conhecimentos adquiridos anteriormente e integrados num todo podem agora ser conceptualizados num nível superior de abstracção e generalização para um novo tópico ou área.

Os níveis da taxonomia SOLO e respectivas descrições encontram-se sumariados na tabela 4.7. Os níveis inferiores da taxonomia SOLO podem ser usados para focar-se em elementos ou atributos individuais relativamente ao que está a ser avaliado. Os níveis mais elevados, com a sua ênfase na integração e na extensão de princípios, requerem uma gama mais ampla de conteúdo referente aos atributos examinados. A taxonomia SOLO não objectiva inferir um nível de processamento cognitivo. No entanto, pode ser argumentado que a obtenção de uma execução a nível Relacional ou superior envolve maior processamento cognitivo do que aquele exigido num nível Uni-estrutural ou Multi-estrutural, desde que os alunos não apenas tenham de recordar elementos, mas também tenham que mostrar o relacionamento entre eles (Relacional) e tirar conclusões (Abstracto).

A taxonomia SOLO é muito diferente das outras taxonomias revistas no trabalho realizado pelo grupo de Fuller et al. (2007), uma vez que lida com o conteúdo da resposta do aluno relativamente ao que está a ser avaliado. A sua aproximação holística significa que pode ser usada para avaliar o desempenho nos domínios afectivo, psicomotor ou cognitivo. Os seus pontos fortes consistem em incentivar uma abordagem de suporte a uma aprendizagem profunda. Biggs (1999) fornece exemplos de estratégias de avaliação para níveis específicos da taxonomia SOLO bem como estratégias mais holísticas. Porém ainda não existem muitos relatos de experiências sobre a sua utilização referente à avaliação de escalas de assuntos.

Nível	Descrição
Pré-estrutural	Conhecimento desgarrado ou plágio de determinada matéria. A informação apresentada tem pouca ou nenhuma relevância face às exigências pedidas.
Uni-estrutural	Concentração num conceito ou nomeação de coisas. Exibição de conhecimento mínimo, tendo em consideração apenas uma das características dos requisitos.
Multi-estrutural	Listagem de elementos mas sem relacionamento entre eles. A ênfase é dada à quantidade e não à qualidade dos conhecimentos.
Relacional	Compreensão através da integração de conceitos e ideias. Compreensão sobre como aplicar o conceito a um problema familiar.
Abstracto	Relacionamento de um conceito ou princípio existente de forma a conseguir resolver problemas novos. Questionamentos e reflexões para além dos princípios existentes.

Tabela 4.7: Níveis da Taxonomia SOLO

4.6 Taxonomia da matriz

Fuller et al. (2007) propõem uma nova taxonomia, adequada, segundo estes autores, às engenharias e ciências da computação e especialmente útil para a aprendizagem da programação. A ideia desta taxonomia é a de fornecer uma estrutura mais prática para avaliar as capacidades dos principiantes nestas áreas. No entanto, esta taxonomia poderá ser vista apenas como uma solução parcial uma vez que não endereça os domínios afectivo e psicomotor, tratando apenas indirectamente as competências de abstracção e, de forma incompleta, as relações estruturais do conteúdo. A inspiração para esta taxonomia residiu nas pesquisas realizadas por Winslow (1996) e Lister et al. (2004) que indicam que a capacidade de compreensão do código de um programa e a capacidade para produzir código são duas aptidões semi-independentes. Os alunos que conseguem interpretar programas podem não conseguir necessariamente escrever os seus próprios programas. Também a capacidade de escrever código não implica necessariamente a capacidade de corrigir erros. Robins et al. (2003) descrevem esta capacidade como a habilidade de distinguir o comportamento pretendido do comportamento real de um programa.

Embora uma revisão da literatura revele alguma diversidade de taxonomias candidatas, apenas a taxonomia de Bloom do domínio cognitivo tem sido amplamente utilizada no planeamento de cursos e avaliação em ciências da computação. As suas principais vantagens residem na facilidade razoável de compreensão dos níveis da

hierarquia, para além da existência de uma vasta literatura em desenvolvimento sobre como usá-la para planeamento de actividades e avaliações. Desta forma, os autores consideraram que essa seria a base mais natural para propor a sua taxonomia, porém acharam mais adequado usarem a versão revista da taxonomia de Bloom. A taxonomia proposta por estes autores, apresentada na tabela 4.8, utiliza então uma matriz bidimensional, resultante da adaptação da taxonomia de Bloom revista, com os nomes dos níveis oriundos dessa taxonomia, que estes autores consideram suficientemente inequívocos.

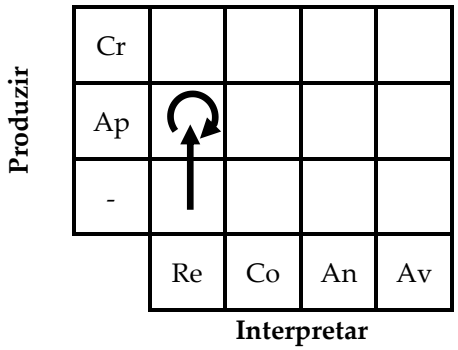
Criar				
Aplicar				
Nenhum				
	Recordar	**Co**mpreender	**An**alisar	**Av**aliar

(adaptada de Fuller et al., 2007)

Tabela 4.8: Adaptação bidimensional da Taxonomia de Bloom

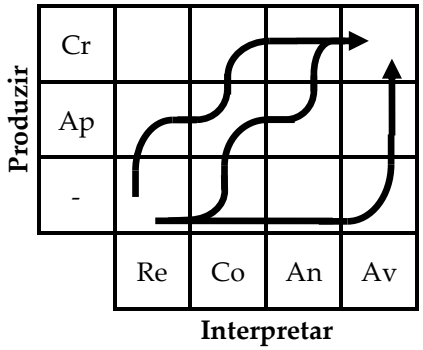
As dimensões da matriz representam as duas escalas de competências de forma separada, nomeadamente a capacidade para compreender e interpretar um produto existente (programa) e a capacidade de projectar e construir um produto novo. Os níveis relativos à compreensão e interpretação de código são colocados no eixo horizontal e os níveis relativos à geração de código são colocados no eixo vertical, ambos com os níveis mais baixos no canto inferior esquerdo. Compreende-se assim que os alunos atravessem cada linha através de uma sequência estrita. Por exemplo, não é possível começar a fazer a síntese (Criar) sem que haja um determinado grau de competência através do nível de aplicação (Aplicar).

A matriz deve ser especialmente útil para os educadores que precisam de uma grelha para classificar os seus alunos. Também ilustra os diferentes percursos de aprendizagem que os alunos podem seguir, como mostrado no trabalho recente realizado por Lahtinen (2007). Por exemplo, quando um aluno aprende um novo conceito de programação consegue primeiramente o conhecimento desse conceito. Nesse ponto o aluno encontra-se na célula (estado de) "Nenhum/Recordar". Se este aluno continuar com uma aprendizagem por imitação de um exemplo de um programa, mas sem a sua compreensão profunda, conseguirá o estado "Aplicar/Recordar", isto é aplicar ou tentar aplicar o conceito sem uma compreensão real, por tentativa e erro. Este comportamento é ilustrado na tabela 4.9.

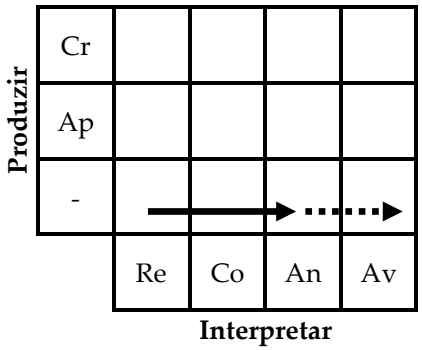


(adaptada de Fuller et al., 2007)
Tabela 4.9: Estado de "Aplicar/Recordar"

Se em vez da imitação, o aluno decidir procurar primeiro mais informação sobre esse conceito, pode prosseguir para a célula "Nenhum/Compreensão" à direita da célula inicial. Isto significa que o aluno não consegue ainda produzir o código do programa, mas já consegue compreender o significado subjacente a esse conceito. Um profissional competente relativamente a determinado conceito seria colocado na célula "Criar/Avaliar", significando que é proficiente em todos os níveis de competência da matriz. Isto pode igualmente ser identificado como o nível de aplicação mais elevada (Johnson e Fuller, 2006) e pode ser alcançado através de diferentes trajectos (tabela 4.10). Porém, há alunos que alcançam somente algumas das competências. Por exemplo, os estudantes teóricos identificados em Lahtinen (2007) podem ser colocados na célula "Nenhum/Avaliação", significando que são capazes de interpretar, analisar e avaliar o código de um programa, mas não conseguem ainda projectar uma solução ou produzir a totalidade do código. Este não é o caminho mais comum que os alunos habitualmente seguem, mas pode acontecer que os alunos prossigam apenas no sentido horizontal como mostrado na tabela 4.11.

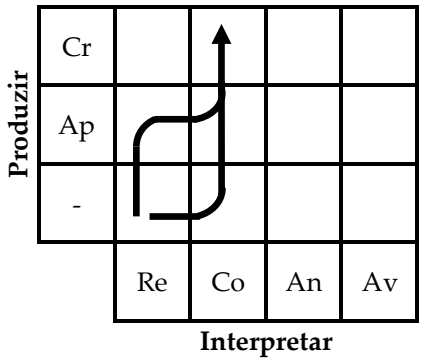


(adaptada de Fuller et al., 2007)
Tabela 4.10: Percursos para o estado "Criar/Avaliar"



(adaptada de Fuller et al., 2007)
Tabela 4.11: Estado de "Nenhum/Avaliar"

O mesmo estudo revelou um outro grupo, designado de estudantes práticos, que poderiam ser colocados na célula "Criar/Compreender" da matriz. Estar nessa célula indica a capacidade de aplicar e sintetizar sem a capacidade de analisar ou avaliar mesmo o seu próprio código. Este comportamento é ilustrado na tabela 4.12. O grande problema para estes estudantes práticos está em não conseguir detectar os erros das suas próprias soluções.



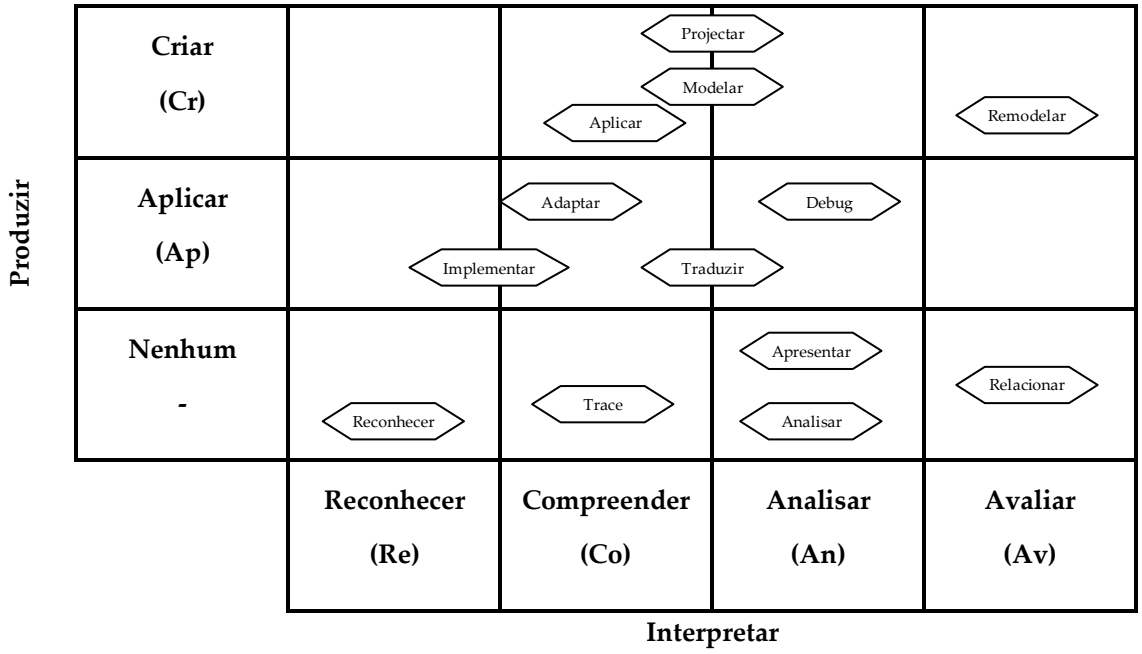
(adaptada de Fuller et al., 2007)
Tabela 4.12: Estado de "Criar/Compreender"

Estes autores estabeleceram um mapeamento entre um conjunto de actividades de programação e as células da matriz, a fim de ilustrar o poder discriminatório da taxonomia proposta para esta área. Desta forma, os autores apresentam uma lista de actividades de programação como ilustrado na tabela 4.13. As actividades mostradas na tabela 4.14 são então mapeadas nas células da taxonomia.

Actividade	Descrição
Adaptar	Modificar uma solução para outros domínios/escalas
Analisar	Examinar a complexidade de uma solução
Aplicar	Usar uma solução como um componente de um problema maior
Apresentar	Explicar uma solução aos outros
Debug	Detectar e corrigir erros
Implementar	Codificar uma solução
Modelar	Ilustrar ou criar uma abstracção de uma solução
Projectar	Planear a estrutura de uma solução
Reconhecer	Conhecer conceitos básicos e vocabulário do domínio
Relacionar	Compreender uma solução no contexto de outras
Remodelar	Optimizar uma solução
Trace	Verificação controlada de uma solução
Traduzir	Traduzir/explicar uma solução

(adaptada de Fuller et al., 2007)

Tabela 4.13: Lista de actividades de resolução de problemas relacionadas com programação



(adaptada de Fuller et al., 2007)

Tabela 4.14: Mapeamento de actividades de programação na taxonomia da matriz

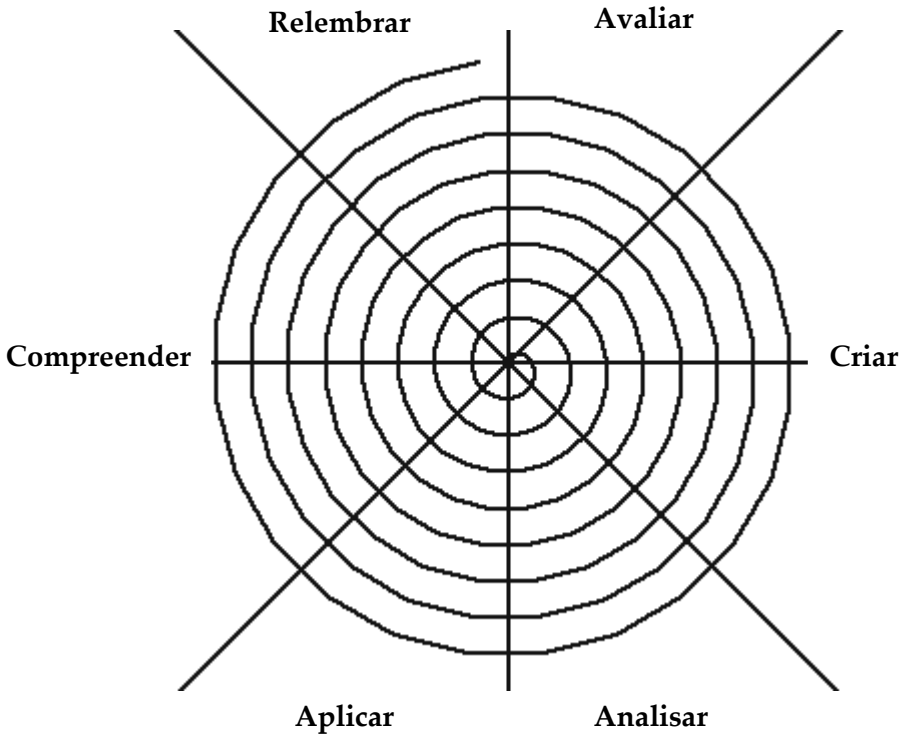
De acordo com estes autores, "Adaptar" uma solução exige provavelmente uma competência próxima de Criar, na escala vertical e pelo menos Compreender na escala horizontal, uma vez que adaptar envolve produzir e saber o quê e como modificar, exige compreensão. "Aplicar", de acordo com o descrito na tabela 4.14 pode ser considerada uma competência tão elevada quanto a exigida pelo nível Criar (do eixo vertical) uma vez que apela à capacidade criativa, mais do que provavelmente a implicada pelo nível Aplicar (do eixo vertical), apesar do seu nome. A posição no eixo horizontal depende da complexidade do problema proposto, mas os autores consideram que exige pelo menos as competências do nível Compreender. "*Debug*" implica uma colaboração quer da interpretação quer da construção, o que significa que deverá implicar um posicionamento elevado em ambos os eixos provavelmente próximo da célula Criar e pelo menos da célula Analisar. A habilidade de "Projectar" implica naturalmente Criar na escala vertical e provavelmente algum grau de interpretação (Compreender) e/ou análise (Analisar) na escala horizontal, embora seja difícil de precisar quanto. "Remodelar" e "Relacionar" são mostrados ao mais alto nível de interpretação porque ambos apelam para uma compreensão profunda do contexto do problema e solução. Os autores encaram a remodelação como envolvendo uma melhoria ou optimização do projecto original, admitindo assim uma colocação possivelmente mais elevada do que o "Projectar" no eixo horizontal. Os autores consideram que a utilização de um raciocínio similar permitirá a colocação na grelha, das restantes actividades. Os autores provaram que, na realidade, o mapeamento é praticável e conduz a uma cobertura razoavelmente completa da grelha.

Outro assunto que gerou alguma discussão neste grupo de trabalho foi o de como aplicar a taxonomia da matriz ao domínio afectivo. Estes autores projectaram esta taxonomia somente para o domínio cognitivo e não para as competências não cognitivas (por exemplo habilidades sociais e emocionais e a adopção de padrões profissionais). As possibilidades consideradas incluiriam a extensão da matriz em uma ou ambas as direcções ou o planeamento de uma matriz companheira. Porém, estes autores consideraram que havia tão pouca experiência na avaliação de valores e atitudes em ciências da computação que seria prematuro a inclusão de tais considerações.

4.7 Arquitectura da Espiral aplicada a uma taxonomia

Robins et al. (2003) consideram que a aprendizagem do aluno passa pela aprendizagem de novos esquemas e a alteração e combinação desses esquemas a fim de produzir esquemas novos e mais abstractos. Estes autores descrevem um esquema como "uma estrutura de pedaços de conhecimento relacionado". Assim, a aprendizagem da programação pode ser considerada como um processo iterativo de aprendizagem de esquemas. Programar é uma habilidade que é aprendida através da construção de nova informação sobre informação mais básica e não à custa de informação desgarrada que é aprendida aqui e acolá. Assim, de certa forma, os pedaços básicos de informação com os quais os alunos se debatem, numa fase inicial, transformam-se em pedaços que se usam na aprendizagem subsequente da matéria seguinte. Tal como no caso de outros modelos de aprendizagem cíclica, como por exemplo, o modelo de estilos de aprendizagem experimental de Kolb, a ideia é prosseguir para um novo nível após cada ciclo.

Segundo os proponentes desta taxonomia, a ideia de uma taxonomia de aprendizagem cognitiva pode também ser usada de forma iterativa, em espiral. Numa fase introdutória, seriam ensinados ao aluno fragmentos de informação muito simples e básicos, bem como sítios onde os aplicar. Quando o aluno está a aprender os conceitos básicos e os assuntos mais simples, está a atravessar a taxonomia relativamente apenas a esse assunto. Em seguida, após ter criado um esquema sobre esse assunto, é guiado para um assunto mais abstracto. Ao examinar esse novo assunto, o aluno está a começar outra vez no nível mais baixo da taxonomia, mas agora usando a matéria anterior como pré-requisito. De acordo com os seus autores, o processo em espiral podia ser aplicado, por exemplo, à taxonomia de Bloom. Quando o aluno está a aprender um novo assunto, os seus pré-requisitos – o material a usar para a construção do novo conhecimento – transformaram-se no seu novo conhecimento básico, embora o aluno possa ter alcançado, de acordo com Bloom o nível Criar ou Avaliar nos assuntos mais básicos. Criar poderia ser descrito como a habilidade de combinar um assunto com outros a fim de construir novas soluções. Isto pode igualmente ser visto quando novas soluções ou assuntos são aprendidos pela construção sobre o conhecimento precedente, integrando-o. Este aspecto pode ser facilmente verificado se considerarmos que os tópicos que são difíceis e exigem uma análise detalhada pelos alunos constituem um mero conhecimento básico para os programadores peritos. A aplicação da Taxonomia de Bloom de forma iterativa é ilustrada na figura 4.2, que constitui um exemplo de aprendizagem em espiral.



(adaptada de Fuller et al., 2007)
Figura 4.2: Representação em espiral da Taxonomia de Bloom

Assim, por exemplo, no início um aluno é ensinado sobre como usar uma estrutura repetitiva (ciclo). Atravessará todos os níveis da taxonomia de Bloom ao aprendê-la. O aluno sabe que um ciclo pode ser usado para fazer um conjunto de iterações; compreende como o ciclo funciona; é capaz de aplicar um ciclo quando lhe é pedido, aprendendo-o eventualmente na sua plenitude. Após ter alcançado os níveis mais elevados, a estrutura de repetição transformar-se-á numa ferramenta que o aluno pode utilizar em actividades de programação subsequentes. Quando o aluno está a aprender a ordenar um *array*, o ciclo já deve ser encarado como um conhecimento básico sobre o qual está a construir um novo conhecimento. Posteriormente, quando o aluno está a tentar implementar uma aplicação mais exigente, usará o conhecimento que já interiorizou sobre *arrays* como parte do seu conhecimento básico, e assim sucessivamente.

A aplicação em espiral de uma taxonomia não está limitada a determinada taxonomia como por exemplo a taxonomia de Bloom. Uma volta na espiral (a aprendizagem de um novo "esquema") poderia ser descrita por qualquer taxonomia apropriada que descreva as competências relativas a determinado assunto. Por exemplo, a taxonomia da matriz anteriormente apresentada poderia também ser aplicada em espiral. Um percurso de aprendizagem do nível elementar "Nenhum/Recordar" para o nível mais elevado "Criar/Avaliar" pode ser visto como uma volta na espiral. Quando se passa para um nível

mais elevado de abstracção, o aluno começa o seu percurso de aprendizagem mais uma vez, a partir do canto inferior esquerdo tabela 4.8. Porém, pode acontecer que, ao tentar subir um nível de abstracção (começar um novo círculo na espiral) o aluno possa não ter alcançado o nível mais elevado "Criar/Avaliar" da aprendizagem que lhe devia servir de base. Para usar as competências adquiridas num nível anterior como conhecimento básico para a etapa seguinte, o aluno precisa de dar mais voltas na espiral, significando um domínio de uma maior abstracção, correspondendo ao posicionamento nas células vizinhas tais como "Criar/Analisar". Pode também acontecer que, estando já o aluno a progredir na etapa seguinte (com um assunto mais abstracto), só agora alcance o estado Criar/Avaliar do nível anterior. Assim os dois círculos seriam, de certa maneira, seguidos em paralelo, durante algum tempo. Se o aluno seguiu um dos trajectos de aprendizagem menos desejáveis, ilustrados na tabela 4.11 e na tabela 4.12 (mais teórico ou mais prático) e fez tentativas para progredir para a etapa seguinte, poderá ter construído o seu conhecimento à custa de bases incorrectas (*misconceptions*), o que poderá traduzir-se em dificuldades, numa etapa posterior.

4.8 Estudos na literatura sobre utilização de taxonomias

Variados artigos exploram a forma como várias taxonomias podem ser aplicadas a tópicos de ciências da computação. Em geral, as diferentes taxonomias foram aplicadas no planeamento de cursos, planeamento de materiais de ensino e avaliação, análise de respostas e medição do progresso dos alunos. Nesta secção, serão revistos alguns desses estudos.

4.8.1 Planeamento de Cursos

Alguns autores propõem a utilização de taxonomias para o planeamento ou avaliação de cursos. Howard et al. (1996) propuseram-se a identificar os objectivos definidos para cada aula de programação e atribui-los a determinado nível da taxonomia de Bloom. O estudo demonstrou que a maioria das suas aulas começava com um número de questões correspondentes ao nível do conhecimento. Porém, muitos outros objectivos correspondiam a um mapeamento mais elevado na taxonomia. Os autores, concluíram que era difícil atingir o nível de Avaliação da taxonomia de Bloom em todas as aulas, mas que seria possível o deslocamento para outros níveis elevados da taxonomia ao longo de uma série de aulas.

Outros autores aplicaram a taxonomia de Bloom para remodelar currículos completos ou algumas unidades curriculares que se encontravam desajustadas. Por exemplo, Reynolds e Fox (1996) definiram um currículo em tecnologias da informação, com base no ACM

Curriculum'91, de forma a incluir novas unidades de conhecimento e descrevê-las através de um enquadramento nos níveis da taxonomia de Bloom.

Sanders e Mueller (2000) propuserem e implementaram uma remodelação do currículo de ciências da computação para que este fosse acessível a todos os estudantes que revelassem as competências mínimas para o seguirem. A ideia foi a de, para além de fazer alterações mínimas face a determinados problemas detectados no curso, alterar as percepções negativas dos alunos relativamente às disciplinas de programação. Assim, a estrutura dos currículos foi mais centrada no ensino de princípios fundamentais e competências. A reformulação essencial foi no sentido de que a matéria e actividades estivessem relacionadas com os níveis mais baixos da taxonomia de Bloom nos primeiros anos e nos anos mais avançados com os níveis mais elevados.

Buck e Stucki (2001) esboçaram uma abordagem pedagógica baseada na taxonomia de Bloom para o desenvolvimento cognitivo. Esta estrutura permitia que os alunos compreendessem os conceitos básicos antes que lhes pedissem para os aplicar. Neste artigo, os autores referem que a tendência natural para ensinar programação é contrária aos modelos do desenvolvimento cognitivo. Referem ainda que a Taxonomia de Bloom pode fornecer um bom suporte para uma pedagogia mais eficaz, em particular poderá ajudar a identificar um conjunto de tópicos, exercícios e tarefas adequadas para as disciplinas introdutórias de programação. Para tal apresentam o JKarelRobot, uma extensão do original "Karel the Robot", para dar sustentação a todos os níveis da taxonomia de Bloom. Com esta ferramenta propõem um conjunto de exercícios cognitivamente adequados aos níveis de desenvolvimento dos alunos tendo como guia a taxonomia dos objectivos educacionais de Bloom. Contextualizado, neste ambiente, os autores enfatizam a progressão hierárquica de um conjunto de capacidades e uma aprendizagem gradual, apresentando exemplos de actividades de programação.

Azuma et al. (2003) apresentaram uma expansão e adequação da taxonomia de Bloom, de forma a ser mais ajustada à disciplina de Engenharia de *Software*, não apenas em termos educacionais, mas também para fins industriais.

Doran e Langan (1995) relatam um projecto sobre a implementação de uma aproximação cognitiva (usando a taxonomia de Bloom) para o planeamento das primeiras disciplinas de programação, usando uma sequência estratégica em espiral. No projecto destas disciplinas os autores definiram micro-objectivos que posteriormente foram mapeados em níveis específicos da taxonomia de Bloom, de forma a melhor alcançar e medir esses objectivos e o progresso dos alunos. Este planeamento incluiu também a redefinição

dos materiais de apoio, a reestruturação das aulas laboratoriais, de acordo com determinado nível cognitivo, bem como o tipo de *feedback* gerado.

Oliver et al. (2004) analisaram a dificuldade cognitiva de seis disciplinas, três das quais eram disciplinas consecutivas de programação em C++ e outras três incidiam sobre Redes e Comunicação de Dados. Os autores discutiram a ideia de uma categorização de Bloom para as diferentes disciplinas. Nesse sentido, analisaram o planeamento e resultado das avaliações, em termos da taxonomia de Bloom, resultando numa métrica de dificuldade a que chamaram de *Bloom Rating*. Esta classificação serviu para verificar o desenvolvimento das disciplinas em termos de exigências cognitivas. Estes autores notaram que alguns módulos iniciais do curso de que se destacam as disciplinas introdutórias de programação, possuíam uma classificação elevada em termos de exigências cognitivas, quando comparados com outros mais avançados (não de programação) presentes no final do curso. Estes resultados permitiram um conjunto de reflexões. Em primeiro lugar, os autores concluíram que o curso deveria permitir desenvolver as habilidades cognitivas dos alunos ao longo dos seus três anos, direccionando os alunos para um nível cognitivo mais baixo no início do grau e trabalhando no sentido de desenvolver níveis mais elevados no seu final.

4.8.2 Planeamento de materiais de ensino e avaliação

Uma outra forma de aplicar estas taxonomias consiste na sua utilização para a concepção de materiais de ensino e avaliação. A sua utilização revela-se útil para a estruturação de materiais para ajudar os alunos a progredir dentro de uma taxonomia ou para a estruturação de materiais de avaliação de forma a avaliar uma ampla escala de níveis de desenvolvimento cognitivo. Variados autores têm discutido como é que as taxonomias de aprendizagem podem ser usadas para conceber avaliações. Por exemplo, Lister (2000) reparou que a abordagem inicial para ensinar programação, bem como as avaliações típicas destas disciplinas "saltam" para os níveis mais elevados da taxonomia de Bloom, ao pedirem aos alunos que escrevam programas completos, desde uma fase inicial. Relativamente ao ensino da programação, este autor dá sugestões de actividades, tendo como guia a utilização de uma taxonomia como a de Bloom. Assim, refere que os níveis mais baixos devem enfatizar a capacidade de interpretação e compreensão de código, os dois níveis intermédios devem centrar-se na escrita de pequenos fragmentos de código, dentro de um contexto bem definido e apenas os dois níveis superiores devem consistir na escrita completa de programas não triviais. Este autor apresenta ainda um projecto de curso e exemplos de avaliações que possibilitem o deslocamento dos alunos pela hierarquia de Bloom. Porém, não trata dos níveis superiores, nomeadamente Síntese e Avaliação, por considerar, que a prática comum de ensino e avaliação estão adequadas a estes níveis e, como tal, nada tem a

acrescentar ou modificar. Especificamente propõe testes de escolha múltipla para a avaliação das capacidades de programação dos níveis intermédios, ao contrário da generalidade das opiniões que encaram este tipo de testes adequados apenas para avaliar perguntas de níveis mais elementares. Relativamente a este assunto, cite-se ainda Farthing et al. (1998) que apresentam um projecto que consiste num novo tipo de questões de escolha múltipla (PMCQs - *Permutational Multiple-Choice Questions*) para avaliar habilidades de mais alto nível.

Scott (2003) referiu que a avaliação devia medir o nível alcançado por cada aluno e a classificação deveria depender da sua realização. O autor refere que muitos testes de programação são desadequados, medindo apenas capacidades extremas, muito baixas ou demasiado elevadas. Salienta ainda que se os testes englobassem questões de todos os níveis, ter-se-ia uma medida mais correcta sobre a aprendizagem e progresso dos alunos. Neste artigo, o autor discute cada uma das categorias da taxonomia de Bloom e dá exemplos de programação adequados a cada uma destas categorias. Posteriormente, o autor, avalia um teste dado no passado, de acordo com as categorias definidas por Bloom, observando que esse teste apenas atingiu os níveis 3 (Aplicação) e 6 (Avaliação) da taxonomia. O autor sumaria então um conjunto de princípios que na sua óptica constituirão um tipo de teste mais adequado para a medição da aprendizagem do aluno.

Lahtinen e Ahoniemi (2005) têm estudado a utilização das taxonomias aplicadas a visualizações com a finalidade de ajudar os alunos a compreender a programação não somente nos níveis cognitivos elementares mas também no sentido de suportar o seu progresso até aos níveis mais elevados. Os seus trabalhos consistiram na análise dos níveis da taxonomia de Bloom e, para cada um deles, sugeriram os tipos de visualizações mais relevantes e adequadas a cada nível, resultando numa categorização de exemplos de visualização de programas.

Também Naps et al. (2003) têm estudado a eficácia educacional das visualizações no ensino da programação. Estes autores identificaram um conjunto de boas práticas para o desenvolvimento de visualizações eficazes, do ponto de vista da aprendizagem (adequada ao nível cognitivo do aluno) e do ensino (conveniente para o professor). Estes autores sugerem a utilização da taxonomia de Bloom como uma estrutura padrão para medir tal eficácia.

Refira-se ainda Hernán-Losada et al. (2004) que descrevem as inseguranças e ambiguidades que encontraram ao aplicar taxonomias à concepção de ferramentas educacionais. Classificaram as dificuldades em duas classes: terminologia e complexidade inerente da própria programação. Propuseram então um guia para usar a taxonomia em ciências da computação. Relativamente a este aspecto, num seu artigo mais recente (Hernán-Losada et al., 2006) descreveram as suas experiências com a concepção e desenvolvimento de

ferramentas de aprendizagem inspiradas pela taxonomia de Bloom. Apresentam uma estrutura genérica para o projecto destas aplicações e descrevem as ferramentas desenvolvidas para a aprendizagem de programação orientada a objectos.

Johnson e Fuller (2006) relatam um estudo realizado sobre os 54 testes de avaliação fornecidos aos alunos do 1º ano de ciências da computação da sua universidade. Esses testes foram examinados por um painel de cinco professores do departamento (alguns dos quais envolvidos nessas disciplinas) a quem foi pedido que se pronunciassem sobre os níveis da taxonomia de Bloom avaliados em cada um dos testes. Uma conclusão obtida com estes estudos foi que o nível mais significativo alcançado na maioria das disciplinas foi o nível Aplicação. Aplicar técnicas para a criação de produtos parecia estar no núcleo das actividades da computação. Porém, para problemas de aplicação complexos é necessário que os alunos usem também habilidades classificadas em níveis da Análise/Síntese/Avaliação. Os autores propuseram então um novo nível de "aplicação superior" para assuntos tais como a programação. Este nível abrangeria a actividade cognitiva necessária para a resolução de um problema, ainda que precise das capacidades tradicionais de mais alto nível que conduzem os alunos a um nível de Análise/Síntese/Avaliação. Na generalidade estas capacidades surgem na literatura referidas como a capacidade de abstracção, referida frequentemente como uma capacidade nuclear para muitas áreas das ciências computacionais, como referido, por exemplo, por Kramer (2007). Acerca deste assunto, este autor discute o modelo do desenvolvimento cognitivo de Piaget, que consiste em quatro estágios: sensorial, pré-operacional, operacional concreto e operacional formal (Piaget e Inhelder, 1969). O seu argumento é baseado nos estudos que mostram que uma percentagem significativa da população não desenvolve este estado final da taxonomia: não progride para o estágio onde tem de utilizar significativamente os processos operacionais formais. Seguindo este raciocínio, defendem que conduzir os alunos para este estágio é uma condição prévia para que os alunos percebam muitos aspectos da computação. Referem ainda que os cursos deviam ser projectados de forma a assegurar que os alunos alcançariam este estágio de desenvolvimento cognitivo ainda antes de lhes ser ensinada a maioria dos tópicos de programação. Realçam ainda a utilidade de usar medidas referentes à capacidade de abstracção como uma ferramenta para seleccionar alunos para cursos de ciências da computação.

4.8.3 Análise de respostas e medição do progresso dos alunos

Os estudos de Buckley e Exton (2003) reviram e utilizaram a taxonomia de Bloom como uma estrutura descritiva adequada para testar o conhecimento dos programadores. Os

autores deste estudo ilustraram a forma como as várias tarefas de programação se mapeiam nos níveis desta hierarquia. Neste artigo é apresentado um estudo com dois estudantes mostrando como é que o seu conhecimento relativamente a programas se pode mapear nos vários níveis desta hierarquia.

Lister e Leaney (2003a, 2003b) referem que a abordagem tradicionalmente usada para ensinar e avaliar as disciplinas introdutórias de programação consiste em todos os alunos tentarem realizar as mesmas tarefas de programação, supondo a existência de alunos médios. Também Michael Kölling no seu *blog* sobre *Thoughts on Programming Education* (Kölling, 2009), elaborou um documento resultante de estudos que o levaram à conclusão de que uma distribuição típica das classificações a programação, por todo o mundo, independentemente dos contextos sociais ou geográficos, consiste em duas "bossas". O mais grave é que este padrão que os autores descobriram agora existe há 10, 20 e 30 anos. O que significa que a utilização de uma abordagem que objective atingir os alunos médios tem a dupla desvantagem de fazer com que os alunos mais fracos não aprendam quase nada e os alunos mais fortes não sejam suficientemente desafiados, perdendo igualmente a oportunidade de aprender. Assim, apresentam uma solução baseada na taxonomia de Bloom que consiste numa abordagem que inclui diferentes tipos de actividades de acordo com as capacidades dos alunos. Basicamente aos alunos mais fracos são atribuídas tarefas que requerem conhecimento e compreensão, traduzidas na capacidade de interpretar e compreender programas. Aos alunos médios são atribuídas as tarefas tradicionalmente usadas em disciplinas deste tipo. Aos alunos mais capazes são atribuídas tarefas "abertas" (que podem ser resolvidas de diversas formas) correspondentes aos níveis mais elevados de Síntese e Avaliação. Na sua abordagem, os autores agruparam os seis níveis da taxonomia em três pares, de modo a que o sucesso em determinado par de capacidades conduzisse à aprovação com a correspondente classificação de A, B ou C. Além disso, identificaram as práticas de avaliação mais adequadas a cada par, a saber, exercícios laboratoriais, exames de resposta "aberta", exames de escolha múltipla, actividades, projectos ou revisão entre pares.

Estas ideias foram aplicadas por Box (2003) que enfatiza, em particular, a forma como as taxonomias podem ser usadas para fornecer meios transparentes para que as tarefas possam ser explicadas aos alunos. Salienta ainda a utilidade desta abordagem para que os alunos possam compreender a sua classificação, percebendo em particular a forma como o seu desempenho se enquadra no progresso total das disciplinas. Este artigo fornece uma orientação detalhada aos professores que considerem utilizar a taxonomia de Bloom para estruturar as suas avaliações. O artigo reporta-se à aplicação de uma estratégia de avaliação formativa, de forma a encorajar os alunos a envolverem-se em aprendizagens profundas,

aumentando a sua responsabilidade na sua aprendizagem e encorajando-os a um estudo contínuo.

Em Burgess (2005) são apresentadas experiências para a classificação das avaliações, assegurando que a classificação reflecte o nível cognitivo do aluno. Assim, a classificação atribuída a uma avaliação depende do nível da taxonomia de Bloom evidenciado pelas respostas do aluno.

Lister et al. (2006) relatam a sua experiência no que concerne à utilização da taxonomia SOLO para descrever as diferenças entre alunos e professores quanto à maneira de resolver pequenos exercícios de interpretação de código. Os dados foram coleccionados sob a forma escrita e respostas "em voz alta" dos alunos (principiantes) e dos professores (peritos), usando perguntas de exames anteriores. Durante a análise, as respostas foram mapeadas para os diferentes níveis da taxonomia SOLO. Das respostas "em voz alta", os autores descobriram que os professores manifestavam a tendência para apresentar respostas relacionais enquanto que os alunos tendiam a apresentar respostas multi-estruturais.

Whalley et al. (2006) conduziram uma investigação multi-institucional sobre as capacidades de interpretação e compreensão de código por programadores inexperientes. O conjunto de questões foi avaliado com recurso a dois instrumentos, nomeadamente as taxonomias de Bloom e SOLO. Nove das perguntas do exercício eram de escolha múltipla e a última pergunta era de resposta livre em que os alunos tinham de fazer uma descrição sobre uma determinada parte de determinada codificação. As conclusões deste estudo foram que a dificuldade destas perguntas se correlacionava fortemente com o seu posicionamento nas taxonomias (a maioria dos alunos conseguiu realizar as perguntas de nível mais baixo, um subconjunto daqueles conseguiu executar as perguntas de um nível mais alto, e apenas um subconjunto destes últimos conseguiu realizar as perguntas de nível mais elevado). Este estudo consistiu também num reforço do estudo conduzido por Oliver et al. (2004) cuja classificação das disciplinas introdutórias e intermédias de programação, de acordo com a taxonomia de Bloom, indicava um nível invariante de dificuldade. Assim, os autores concluíram que o nível de dificuldade dos testes das disciplinas introdutórias de programação, apresentava uma barreira significativa e possivelmente injusta e impeditiva do sucesso dos alunos.

Podem ainda citar-se outros exemplos que relatam a utilização da taxonomia de Bloom. Por exemplo, em Cukierman e Thompson (2007) são apresentadas sugestões com o intuito de ajudar os alunos a planear estratégias de aprendizagem em tópicos de ciências da computação.

4.9 Conclusões

Existe uma grande diversidade de taxonomias, com diferentes propósitos no que respeita à aprendizagem e abrangendo diversos domínios, nomeadamente cognitivo, afectivo e psicomotor. Há autores que consideram que um dos problemas das taxonomias educacionais conhecidas é o facto de serem genéricas e assentarem na suposição de que a hierarquia dos resultados de aprendizagem é a mesma para todos os assuntos, desde História da Arte a Zoologia (Fuller et al., 2007). Outros consideram que assuntos diferentes podem colocar a tónica em diferentes níveis da hierarquia. Por exemplo, em assuntos aplicados como a computação, um objectivo de aprendizagem principal poderá consistir na capacidade de desenvolver produtos (programas). Por outro lado, professores de outras matérias (tais como Literatura) poderão colocar mais ênfase em capacidades de crítica e menos na produção (por exemplo, de narrativas).

Porém, independentemente de algumas divergências, a taxonomia mais amplamente utilizada é a proposta por Bloom. As suas grandes vantagens residem na sua simplicidade e na identificação de aspectos distintos e reconhecíveis do domínio cognitivo. As fraquezas desta taxonomia residem no facto de as categorias nem sempre se revelarem fáceis de aplicar, podendo existir sobreposição entre elas, para além de alguma controvérsia sobre o seu posicionamento na hierarquia relativamente a diferentes assuntos, o que na perspectiva de muitos professores traduz algumas limitações à sua aplicação (Johnson e Fuller, 2006).

Apesar da utilização de uma taxonomia poder servir diversos propósitos educativos a nossa preocupação é a sua utilização para melhor diagnosticar as dificuldades dos alunos. Nesta perspectiva, de todas as taxonomias estudadas, consideramos que a taxonomia de Bloom poderá servir como uma boa referência para a definição de actividades de programação de forma a melhor identificar as dificuldades dos alunos.

Sendo assim, recomendamos que os professores de programação estejam cientes desta problemática e comecem a utilizar uma taxonomia de forma a encaminhar os seus alunos no sentido de perceber quais as suas reais dificuldades orientando-os para actividades de estudo adequadas. O posicionamento exacto das questões em determinada taxonomia é um problema que, na nossa óptica, deverá ser relegado para segundo plano, e a existência de três níveis poderá ser suficiente. Concordamos com Lister quando afirma que "relativamente ao ensino da programação elementar: os dois níveis mais baixos devem enfatizar a habilidade de leitura e compreensão de código, os dois níveis intermédios devem enfatizar a escrita de pequenos fragmentos de código, mas dentro de um contexto bem definido, e os dois níveis superiores devem enfatizar a escrita de programas não triviais completos" e "os estudantes devem primeiramente ser ensinados a interpretar programas antes de os escrever" (Lister,

2000). Não queremos com isto significar que se diminua o nível de dificuldade das avaliações, cremos até que uma verdadeira avaliação em programação deverá consistir na escrita de programas completos, integradores de diversos conceitos e saberes.

Cap. 5

Ferramentas de apoio ao ensino e aprendizagem de programação

"A minha função é mostrar o caminho e as ferramentas necessárias, mas o que motiva é a tua acção!"

César Lustosa

5.1 Introdução

As linguagens de programação normalmente utilizadas para aprender a programar apresentam características muito próprias com sintaxes extensas, não facilmente apreensíveis num estágio inicial de aprendizagem de programação, para além de possuírem determinados aspectos que exigem um elevado grau de abstracção. Também os ambientes utilizados para programar são mais adequados para fins profissionais do que para objectivos pedagógicos. A tentativa de fazer um programa e compilá-lo é um dos primeiros problemas com que os alunos se defrontam. As opções são geralmente numerosas e a configuração para a sua utilização não é, geralmente, a mais intuitiva. Quando o aluno consegue compilar um programa, depara-se com outro conjunto de dificuldades. As mensagens de erro são normalmente pouco informativas e confusas. Outro problema refere-se ao facto de os *debuggers* serem normalmente muito complexos necessitando que os alunos entendam conceitos avançados antes de serem realmente úteis. Contudo, pensamos que o maior défice destes ambientes é a não inclusão de representações visuais ou mesmo animações dos algoritmos construídos e implementados.

A importância das representações visuais na compreensão dos programas computacionais não constitui novidade. Goldstein e von Newmann (1947) demonstraram a

utilidade dos diagramas de fluxo, enquanto que Haibt (1959) desenvolveu um sistema que podia desenhá-los automaticamente a partir de programas em linguagens como o Fortran ou o Assembly. Knuth (1963) desenvolveu um sistema similar com documentação integrada com o código fonte, que podia automaticamente gerar diagramas de fluxo. Scanlan (1989) apresentou resultados sobre a importância dos diagramas de fluxo como um contributo para a compreensão de programas.

Uma aproximação diferente foi seguida com os filmes de Knowlton (1966a, 1966b), constituindo o primeiro trabalho a usar técnicas dinâmicas em oposição às técnicas estáticas anteriormente utilizadas e o primeiro a conter visualização das estruturas de dados. Outra abordagem deste tipo foi o *debugger* de Baecker (1968) para o computador TX-2 que produzia imagens estáticas a partir de informação contida num determinado ficheiro, permitindo alguma interactividade. Baecker continuou este trabalho numa direcção mais pedagógica resultando em sistemas para mostrar abstracções das estruturas de dados em programas em execução (Baecker, 1975) e posteriormente no filme *Sorting Out Sorting* (Baecker, 1981).

Os anos 70 assistiram a um regresso aos diagramas de fluxo com o desenvolvimento dos diagramas de Nassi-Schneiderman (Nassi e Schneiderman, 1973) para contrariar a natureza desestruturada dos tradicionais diagramas de fluxo. Roy e St-Denis (1976) desenvolveram então um sistema para gerar automaticamente diagramas de Nassi-Schneiderman a partir de código fonte. Os restantes desenvolvimentos a partir de 1970 debruçaram-se mais no sentido de possibilitar uma leitura mais fácil do código fonte numa linguagem estruturada, recorrendo a técnicas como o uso de espaçamento, indentações, entre outros aspectos que melhoravam a legibilidade de um programa. Foram então desenvolvidos alguns sistemas automáticos, tais como NEATER2 (Conrow e Smith, 1970) para PL/I (*Programming Language One*) e os sistemas de Hueras e Ledgard (1977) para Pascal.

Nos anos 80 iniciou-se a visualização de programas com a introdução de tecnologias de imagens de *bitmaps* e de interfaces de janelas. Um dos sistemas mais importantes e bem conhecido dessa época foi o BALSA (Brown e Sedgewick, 1985), seguido pelo BALSA-II (Brown, 1988), que permitiam que os estudantes interagissem com visualizações dinâmicas de programas em Pascal. Desde então têm sido desenvolvidos vários protótipos e sistemas usando as tecnologias mais recentes.

A diversidade de sistemas originou vários levantamentos taxonómicos sobre sistemas de visualização de programas. Uma importante e conhecida taxonomia é a de Myers (1986) que foi actualizada duas vezes (Myers et al., 1988; Myers, 1990). Neste seu primeiro artigo, Myers forneceu uma diferenciação entre programação por exemplos, programação visual e visualização de programas. Neste seu último artigo, debruçou-se sobre 19 sistemas de

visualização de programas e classificou-os segundo dois eixos: o seu nível de abstracção (podendo mostrar pequenas partes de código, dados, algoritmos ou programas completos) e o nível de animação das suas exibições (estilo da amostragem: estática ou dinâmica). Stasko e Patterson (1992) defenderam uma taxonomia de quatro categorias, sendo cada sistema classificado de acordo com aspectos cobertos, abstracção, animação e automação. Também Shu (1988) se concentrou nos graus crescentes de sofisticação exibida pelos sistemas de visualização de programas. Chang (1990), apesar de não fornecer uma taxonomia, caracterizou a visualização de programas como o uso de representações gráficas para melhor ilustrar os programas, os dados, a estrutura ou o comportamento dinâmico de um sistema. Brown e Sedgewick (1985) propuseram uma classificação de animações de algoritmos ao longo de três eixos: conteúdo ou assunto, transformação (alteração discreta ou continua das imagens) e persistência (representações do estado corrente ou da história completa de execução).

Roman e Cox (1989) sugeriram uma taxonomia baseada num tratamento da visualização como um mapeamento de programas em representações gráficas. Com esta aproximação foi possível caracterizar as visualizações de acordo com o seu domínio (a visibilidade da informação que é extraída), a sua gama, os meios pelos quais o mapeamento é especificado e o nível de abstracção realizado pelo processo de mapeamento (quais as técnicas gráficas usadas). Esta definição também sugere que o processo de visualização seja o resultado de uma acção recíproca entre três participantes: o programador que desenvolve o programa original, o animador que define e constrói o mapeamento e o utilizador que observa a representação gráfica. Os autores examinaram, com mais detalhe, um dos critérios taxonómicos por eles definidos, o nível de abstracção usado pela visualização. Assim, explicam como é que diversos níveis de abstracção poderiam ser aplicados a um algoritmo para o ordenamento de um *array*.

Em 2005, foi apresentada uma taxonomia de linguagens e ambientes de programação projectados de forma a tornar a programação mais acessível aos iniciantes de programação de todas as idades (Kelleher e Pausch, 2005). A classificação é complexa e nela são incluídas ferramentas com diferentes propósitos, por exemplo, para expressar programas simplificando o código, através da redução da linguagem usada ou utilizando mecanismos para impedir erros de sintaxe. Outras abordagens consistem em alternativas à escrita tradicional de programas. Nestas abordagens, especialmente adequadas para públicos mais jovens, a construção de programas é feita usando objectos gráficos ou físicos ou usando determinadas acções da interface. Outros sistemas, tentam fazer com que a programação seja mais concreta, utilizando actores em micromundos. Outros ainda foram concebidos com o intuito de permitir uma aprendizagem social, lado a lado, com pares, uma interacção através

da rede ou fornecendo um contexto motivador. Outros pretendem tornar determinada linguagem de programação mais compreensível, melhorando os mecanismos de interacção, a integração com o ambiente ou a inclusão de sistemas de informação adicional. Passamos de seguida a apresentar alguns sistemas que consideramos mais representativos para a área em estudo.

5.2 Micromundos

De enorme reputação e utilização destacam-se os micromundos. Os micromundos, também designados como mundos programáveis, têm como objectivo possibilitar ao aluno a interiorização de conceitos básicos de programação através de um ambiente familiar e agradável, permitindo especificar tarefas básicas como a movimentação de um personagem e observar os resultados dessa especificação. Desta forma, pretende-se que o aluno utilize conceitos de programação num ambiente mais familiar e concreto. Estes ambientes utilizam normalmente uma minilinguagem que permite suportar os primeiros passos na aprendizagem de programação. As minilinguagens foram fortemente inspiradas no LOGO desenvolvido por Papert (1980). O LOGO não foi concebido especificamente com o propósito de ensinar programação, mas surge como um bom instrumento para introduzir conceitos de programação a alunos sem experiência neste domínio. Esta linguagem permite controlar os movimentos de uma entidade, geralmente uma tartaruga, à custa de comandos básicos e mecanismos de programação simples. Trata-se de uma linguagem que permite introduzir conceitos de programação a alunos sem qualquer experiência neste domínio, especialmente os de escalão etário mais baixo (Schaub, 2000).

O primeiro e mais popular dos micromundos foi "Karel the Robot" (Pattis, 1981), criado com o intuito de auxiliar os alunos a aprender programação em Pascal. Karel é um robô que vive num mundo muito simples e reduzido, consistindo basicamente em avenidas, de norte a sul e ruas, de leste a oeste. Existem, adicionalmente paredes que permitem bloquear as avenidas e ruas, podendo ser colocados sinalizadores na sua intersecção. O robô pode mover-se pelo ambiente a menos que o caminho esteja bloqueado por uma parede. Além de simples translações, o robô pode efectuar rotações e deixar ou apanhar sinalizadores. Através de um simulador os alunos podem ver passo-a-passo o progresso dos seus programas, sendo possibilitado neste sistema a definição de procedimentos e novas instruções recorrendo às existentes.

Posteriormente, surgiram vários ambientes inspirados no Karel, usando algumas das suas características. Martino (Olimpo et al., 1985), Marta (Calabrese, 1989), Pascal Genie

(Chandhok e Miller, 1989), Darel (Kay e Tyler, 1992) e Karel Genie (Miller et al., 1994) são disso exemplo. Josef the Robot (Tomek, 1982), Robot Brothers (Olimpo, 1988), Playground (Fenton e Beck, 1989), Turingal (Brusilovsky, 1991), Gravitas (Sellman, 1992), Tortoise (Brusilovsky, 1993), KidSim (Smith et al., 1994) ou TurtleGraph (Jehng et al., 1994), são exemplos de outras minilinguagens ou subconjuntos de linguagens que surgiram com o intuito de contribuir para uma melhoria da compreensão e realização das tarefas de programação.

À medida que surgiram novas linguagens e paradigmas, surgiram também novas versões do ambiente "Karel the Robot", como o Karel-3D (Hvorecky, 1992), uma extensão do Karel num mundo 3D ou o Karel++ (Bergin, 1997) para suportar a aprendizagem de conceitos orientados a objectos. JKarelRobot (Buck e Stucki, 2001) é uma das suas mais recentes evoluções, sendo independente da plataforma (escrito em Java), independente do paradigma ou linguagem, e suportando linguagens como o Pascal, Java ou Lisp.

A figura 5.1 apresenta o ambiente geral do JKarelRobot, onde está a ser executado um programa simples consistindo na movimentação do robô de forma a apanhar os sinalizadores. Este exemplo apresenta comandos simples, como a movimentação do robô, através do comando *mover*, o apanhar um sinalizador, com *apanhar-beeper*, recorrendo a estruturas de controlo como ciclos para, por exemplo, apanhar todos os sinalizadores num determinado local.

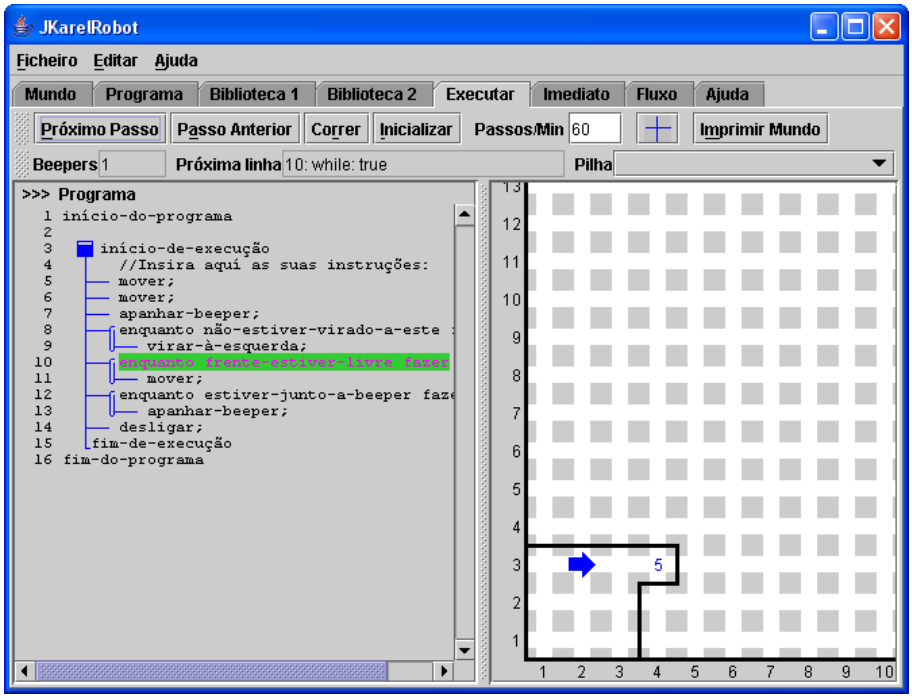


Figura 5.1: Interface do JKarelRobot

Alice constitui outro interessante exemplo de um micromundo muito conhecido para suportar a aprendizagem de programação orientada a objectos, através da criação de mundos 3D (Cooper et al., 2000). No Alice, um ambiente de animação interactivo 3D, o aluno pode aprender a construir e fazer o *debug* de programas. O aluno pode criar o seu próprio mundo, usando modelos de objectos em 3D (animais ou veículos) que povoam o mundo virtual de Alice, sendo possível controlar o seu comportamento e a sua aparência. Um sucessor de Alice é o Alice2 (Kelleher et al., 2002) que usa a ideia base do Alice, mas tem uma interface de criação de programas diferente. Esta nova versão, surgiu após a avaliação do Alice revelar que a sua utilização era dificultada pela escrita dos programas, não muito adequada para alunos num estágio inicial de aprendizagem de programação. Nesse sentido, o Alice2 tenta resolver esse problema, dispondo de um ambiente em que não é necessário digitar o código do programa, mas apenas seleccionar elementos com os nomes dos comandos e dos objectos. Basicamente o código é criado através da selecção de opções e seu arrastamento para determinada posição. Como as linhas de código não podem ser editadas e os arrastamentos e colocações podem apenas ocorrer se estiverem sintacticamente correctos, não surgem erros de sintaxe e dessa forma os alunos podem construir programas sem erros de compilação. Esta ferramenta tem o intuito de reduzir a dificuldade de compreensão dos principais conceitos de programação orientada a objectos (métodos, objectos, herança) pelos alunos iniciantes. Para isso, apresenta a visualização dos objectos num contexto similar ao mundo real. Proporciona um ambiente virtual em que os alunos podem usar e modificar o comportamento dos objectos 3D, sendo possível a criação de novos objectos a partir de objectos básicos. Alice oculta do utilizador os detalhes envolvidos na criação de um programa real, resultando numa ferramenta conveniente para aprendizes com pouco ou nenhum conhecimento de programação. Por meio de uma interface que se assemelha a ambientes IDE para programação real, o utilizador cria e manipula objetos (animais, casas, ferramentas, cenários, entre outros) com o intuito de construir um mundo virtual animado. A interface do Alice2 é ilustrada na figura 5.2.

(1) Cena; (2) Árvore de objectos; (3) Área de detalhe dos objectos;
(4) Área de animação; (5) Área de comportamentos

Figura 5.2: Interface do Alice2

O Scratch[4] é outra ferramenta em desenvolvimento pelo grupo de investigação *Lifelong Kindergarten* do *MIT Media Lab*, em colaboração com o *KIDS research group* do *UCLA Graduate School of Education & Information Studies*. É um "ambiente de programação" que permite facilmente criar histórias interactivas, jogos, animações, entre outros, com diferentes graus de complexidade. Estas criações podem ainda ser partilhadas na *world wide web*. A interface geral pode ser visualizada na figura 5.3 e na figura 5.4. Com este micromundo é possível atingir diferentes escalões etários, produzindo actividades muito diferenciadas adequadas desde o pré-escolar ao ensino universitário, destinadas ao ensino introdutório de programação.

[4] Disponível em http://scratch.mit.edu/

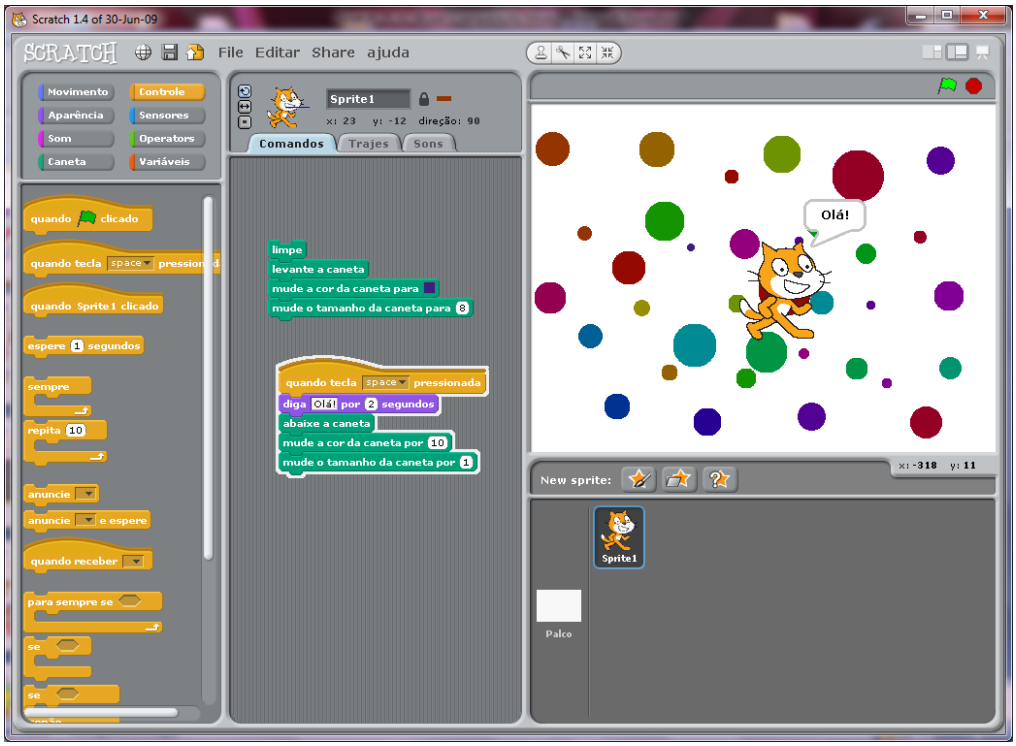


Figura 5.3: Interface do Scratch – Exemplo de uma actividade elementar

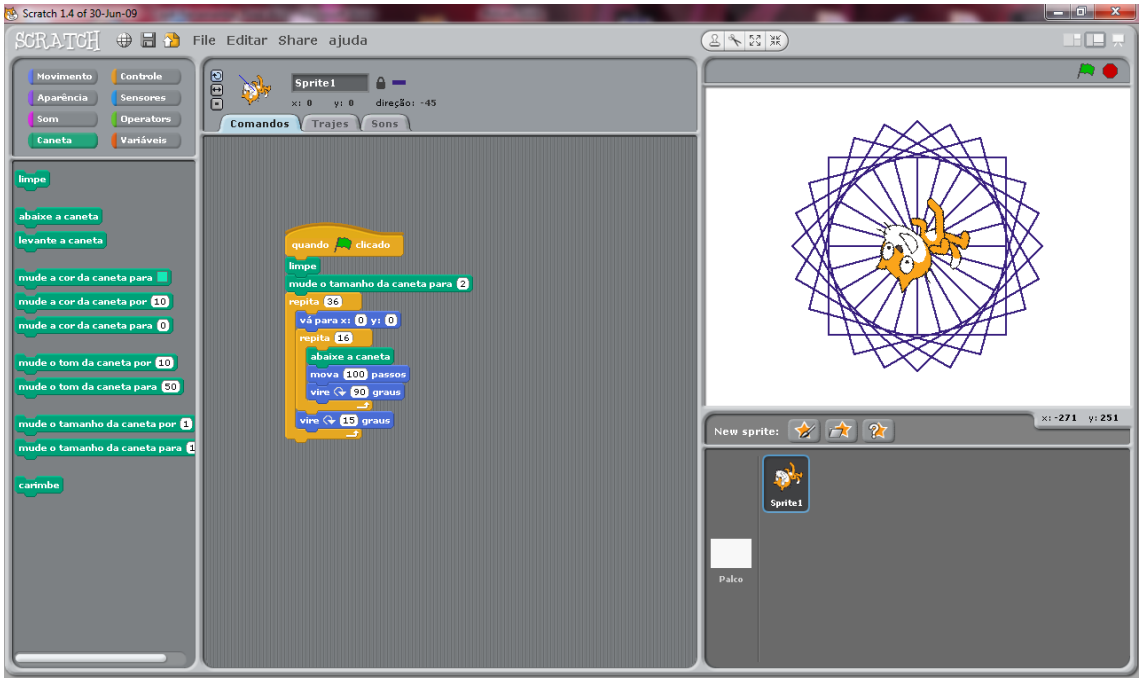


Figura 5.4: Interface do Scratch – Exemplo de uma actividade mais elaborada

5.3 Sistemas de animação e simulação de algoritmos e programas

Os sistemas de animação e simulação de algoritmos e programas têm conseguido captar mais atenções, do ponto de vista dos utilizadores e investigadores. A importância destes tópicos pode ser averiguada pela quantidade de conferências, *workshops* e grupos de trabalho destinados a esta temática. Veja-se, por exemplo, o *workshop* bianual em *Program Visualization*, uma frutuosa actividade para investigadores nesta área (PVW 2000, PVW 2002, PVW 2004, PVW 2006). Valiosos são também os grupos de trabalho existentes nas conferências ITiCSE, podendo citar-se os seguintes:

- 1996: *An overview of visualization: its use and design* (Bergin et al., 1996).
- 1997: *Using the WWW as the delivery mechanism for interactive, visualization-based instructional modules* (Naps et al., 1997).
- 2002: *Exploring the role of visualization and engagement in computer science education* (Naps et al., 2002).
- 2003: *Evaluating the educational impact of visualization* (Naps et al., 2003).
- 2006: *Merging interactive visualizations with hypertextbooks and course management* (Rössling et al., 2006).

Como actividades mais gerais pode também citar-se a *Association for Computing Machinery Symposium on Software Visualization*[5].

O Incense de Myers (Myers, 1980; Myers, 1983) foi o primeiro protótipo de visualização de programas, permitindo criar automaticamente imagens gráficas das estruturas de dados de um programa. Posteriormente Myers e os seus colegas da *Carnegie-Mellon University* criaram um sistema designado Amethyst (Myers et al., 1988) que foi integrado com o ambiente de programação MacGnome Pascal conhecido como Pascal Genie (Chandhok et al., 1991).

O BALSA (Brown e Sedgewick, 1985) foi um dos primeiros sistemas interactivos de animação de algoritmos. Também bastante interessante foi o sistema de animação de algoritmos Zeus (Brown, 1991). Podem citar-se outros sistemas que recorrem a representações visuais e/ou animações, nomeadamente: SEE (Baecker e Marcus, 1986), BALSA-II (Brown, 1988), VIP (Mendes e Mendes, 1988), GAIGS (Naps, 1990), TANGO

[5] Disponível em http://www.st.uni-trier.de/~diehl/softvis/org/softvis06/

(Stasko, 1990), ANIM (Bentley e Kerninghan, 1991), Pavane (Roman et al., 1992), XTANGO (Stasko, 1992), Polka (Stasko e Kraemer, 1993), DRUIDS (Whale, 1994), FLAIR (Ingargiola et al., 1994), Polka-RC (Stasko e McCrickard, 1995), MRUDS (Hanciles et al., 1997), "*The Sort Animator*" (Dershem e Brummund, 1998), Alma (Varanda e Henriques, 1999), entre outros.

Como referido, os sistemas de animação e simulação de algoritmos e programas têm sido muito utilizados com o propósito de facilitar a compreensão de alguns aspectos de programação. Alguns destes sistemas têm um âmbito mais restrito, mostrando apenas manipulações das estruturas de dados ou limitando-se a permitir a visualização de algoritmos pré-programados para áreas de programação específicas, como por exemplo a ilustração de algoritmos de ordenação ou o funcionamento de estruturas como as listas ligadas, árvores binárias ou grafos. Este tipo de animações permite essencialmente a visualização de forma animada da evolução dos dados e/ou estruturas de dados durante a execução do algoritmo. Outros, possuem um nível mais global, mostrando o seu propósito e metodologias, permitindo a animação de pseudocódigo, fluxogramas ou mesmo de programas escritos em linguagens específicas como Pascal, C, Lisp, Java, entre outras. De entre estes resultam particularmente interessantes, aqueles que permitem que os alunos introduzam e simulem os seus próprios algoritmos e programas.

Na categoria de animações mais específicas, podemos indicar alguns sistemas facilmente encontrados na *world wide web*. Um desses exemplos é o *xSortLab*[6], um *applet* desenvolvido por David Eck. Neste sistema o aluno pode visualizar o funcionamento de um conjunto de algoritmos de ordenação, como o exemplo apresentado na figura 5.5. O aluno pode, entre outras coisas, controlar a velocidade da animação e aceder a uma explicação da mesma.

[6] Disponível em http://math.hws.edu/TMCM/java/xSortLab/

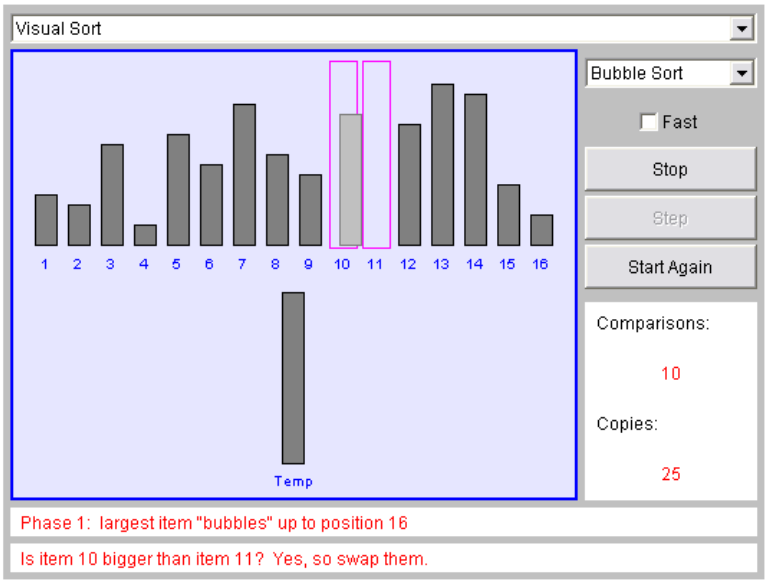


Figura 5.5: Interface do xSortLab – Exemplo de animação do BubbleSort

Na categoria de animações de programas pré-definidos, podemos citar por exemplo o Program Animator[7]. É um programa que permite visualizar a execução de programas escritos em Pascal. O aluno pode executar o programa na totalidade ou passo-a-passo, tal como mostrado na figura 5.6.

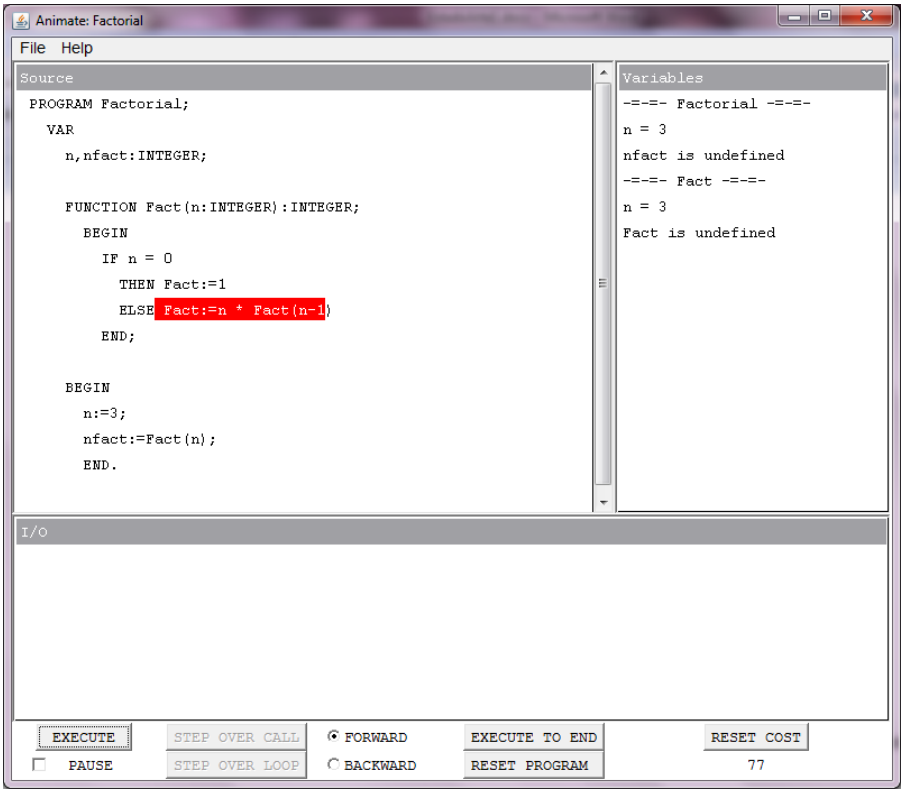


Figura 5.6: Interface do Program Animator

[7] Disponível em http://www.cs.montana.edu/webworks/webworks-home/projects/program_animator/

Um outro exemplo deste tipo é o *Programming Education System Based on Program Animation*[8] (Miyadera et al., 2000). Neste sistema o aluno pode visualizar programas pré-construídos essencialmente sobre algoritmos de ordenação, árvores binárias e listas ligadas. Neste caso, para além de mostrar a animação, é também apresentado o respectivo código em linguagem C, sendo as instruções pertinentes realçadas à medida que a animação vai avançando. Um exemplo de uma animação, neste sistema, é apresentado na figura 5.7.

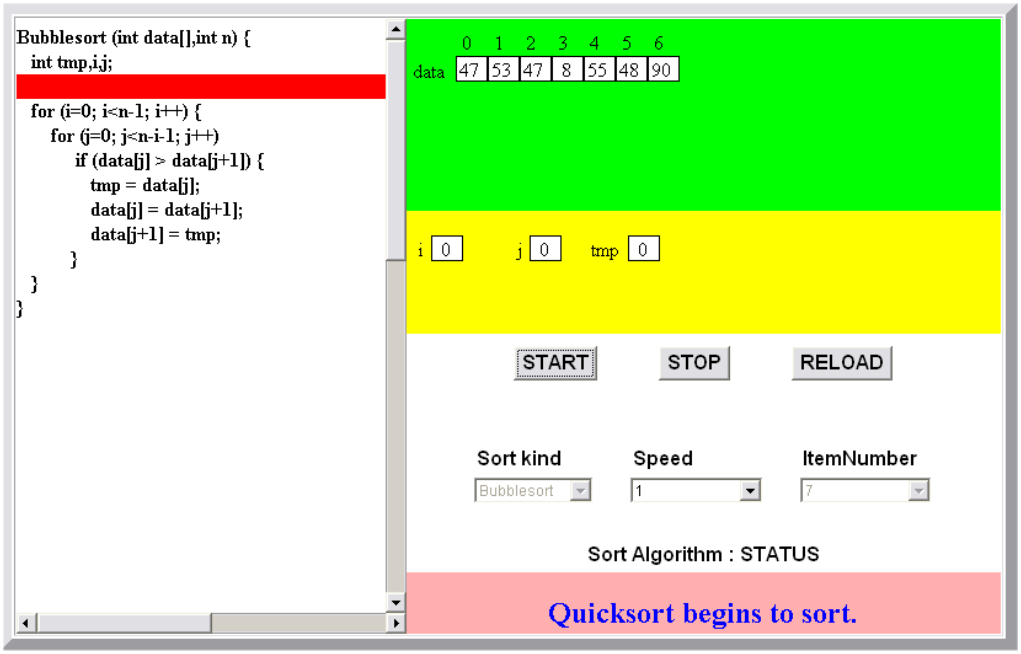


Figura 5.7: Interface do Programming Education System Based on Program Animation

De considerável interesse, nesta categoria destacamos ainda o Jhavé[9] (Naps, 2005) que apesar de permitir apenas a animação de algoritmos pré-definidos, apresenta uma maior diversificação de algoritmos, relativamente aos sistemas anteriormente referidos. Adicionalmente, possibilita um nível de interacção superior com o utilizador, na medida em que vai colocando perguntas sobre o comportamento da animação ao longo da execução do algoritmo. A interface deste sistema é apresentada na figura 5.8.

[8] Disponível em http://www.u-gakugei.ac.jp/~miyadera/Education/system.html
[9] Disponível em http://jhave.org/

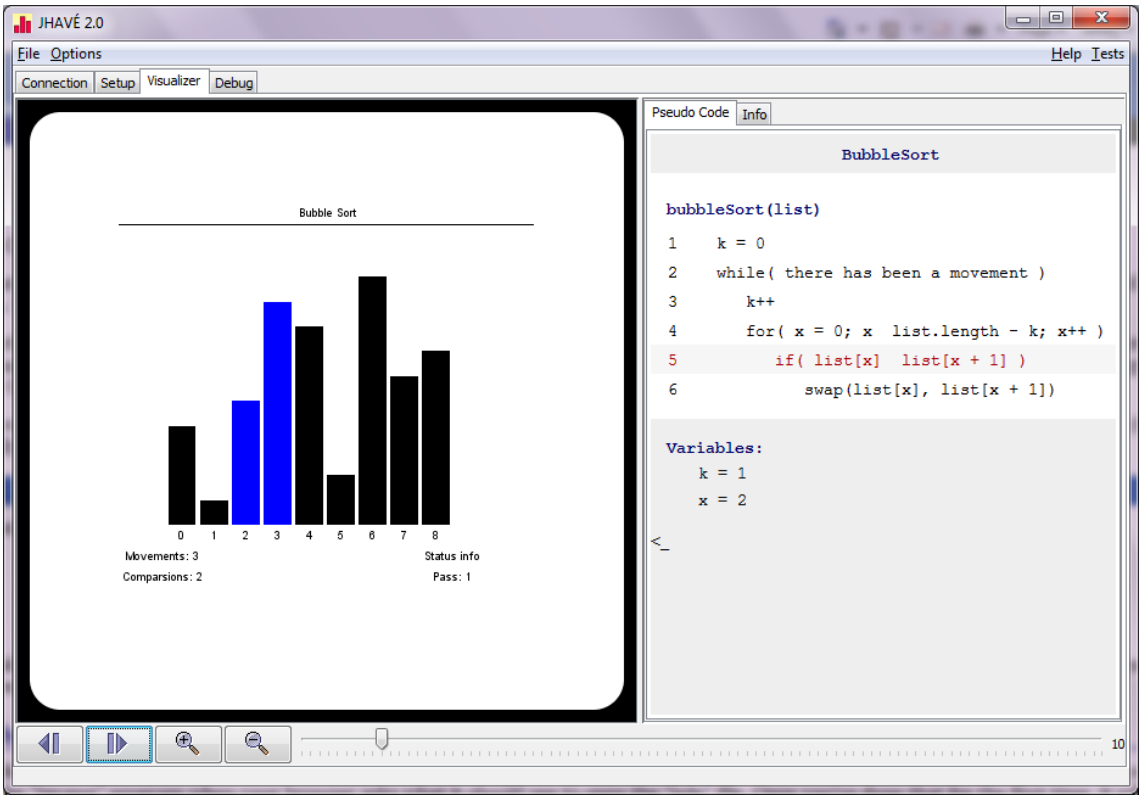


Figura 5.8: Interface do Jhavé

Existem, porém, ambientes que permitem ao aluno simular os seus próprios programas, sendo-lhes permitido visualizar a execução das suas instruções e as respectivas consequências. Alguns destes ambientes permitem animar programas escritos numa linguagem de programação específica, outros recorrem a algoritmos especificados através de pseudocódigo ou representações gráficas, como por exemplo fluxogramas. Um exemplo desses ambientes é o Jeliot 2000[10] (Levy et al., 2003) que se destina ao apoio da aprendizagem de conceitos de programação orientada a objectos. Este sistema interpreta código escrito em Java permitindo gerar a sua animação. É um sucessor do sistema Eliot (Lahtinen et al., 1998), desenvolvido para animar estruturas de dados de programas escritos em linguagem C. De referir também o Jeliot 3 (Moreno et al., 2004), uma evolução do Jeliot 2000. A interface deste ambiente encontra-se ilustrada na figura 5.9, sendo visível uma animação de um programa que faz o cálculo da média de N notas de um aluno.

[10] Disponível em http://www.cs.joensuu.fi/jeliot/

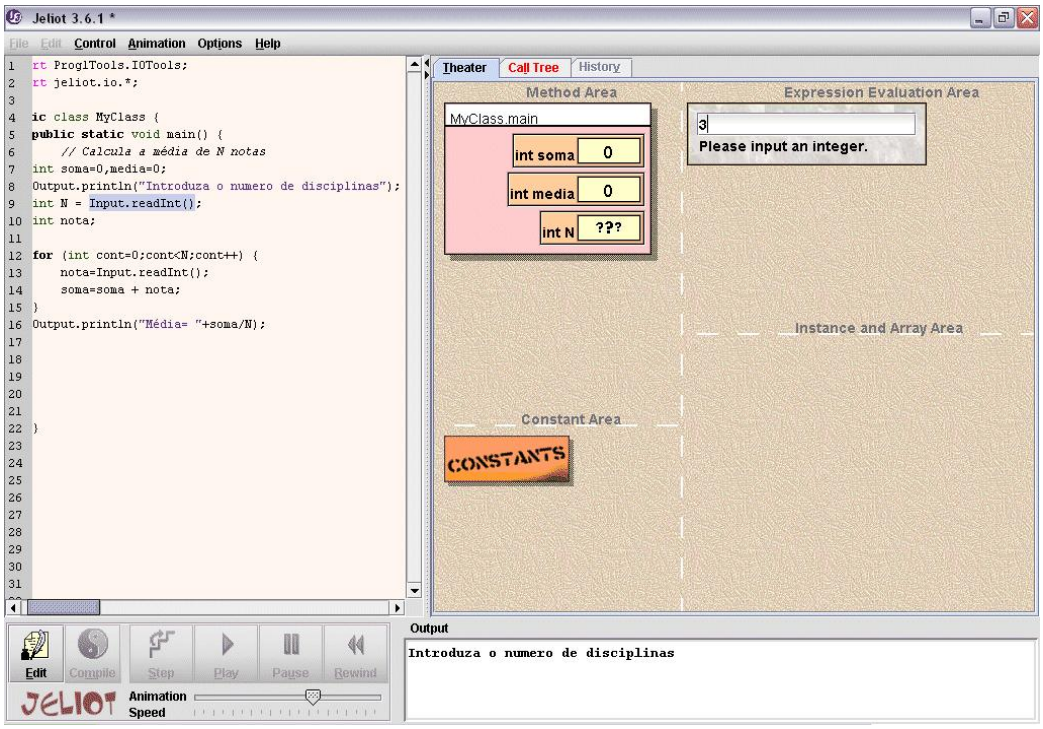


Figura 5.9: Interface do Jeliot3

Cite-se também o PlanAni[11] (Sajaniemi e Kuittinen, 2003), um animador de programas baseado no conceito de papéis/funções das variáveis. Os autores desta ideia defendem a utilização de uma ferramenta que tenha por base estes conceitos (papéis das variáveis) para facilitar a formação de esquemas (que os peritos normalmente utilizam) para uma melhor aprendizagem de programação. De acordo com Sajaniemi (2002), os papéis das variáveis representam estereótipos de utilização das variáveis nos programas. Assim, por exemplo, a uma variável com o papel *Stepper* são atribuídos valores de uma ordem sistemática e previsível (ex: ordem crescente de inteiros: 0, 1, 2, …) enquanto que um valor fixo (*Fixed value*) é uma variável cujo valor nunca muda. No PlanAnim, a cada função/papel de uma variável corresponde uma visualização - imagem da função ou papel da variável - que é usada para todas as variáveis com a mesma função. Para cada visualização são utilizadas metáforas que os seus autores consideram significativas, pois as imagens das funções dão indícios de como os valores sucessivos da variável se relacionam entre eles e com as outras variáveis. Por exemplo, um valor fixo é descrito por uma pedra que dá a impressão de um valor que não é fácil de mudar. A figura 5.10 mostra a interface deste sistema.

[11] Disponível em http://www.cs.joensuu.fi/~saja/var_roles/planani/index.html

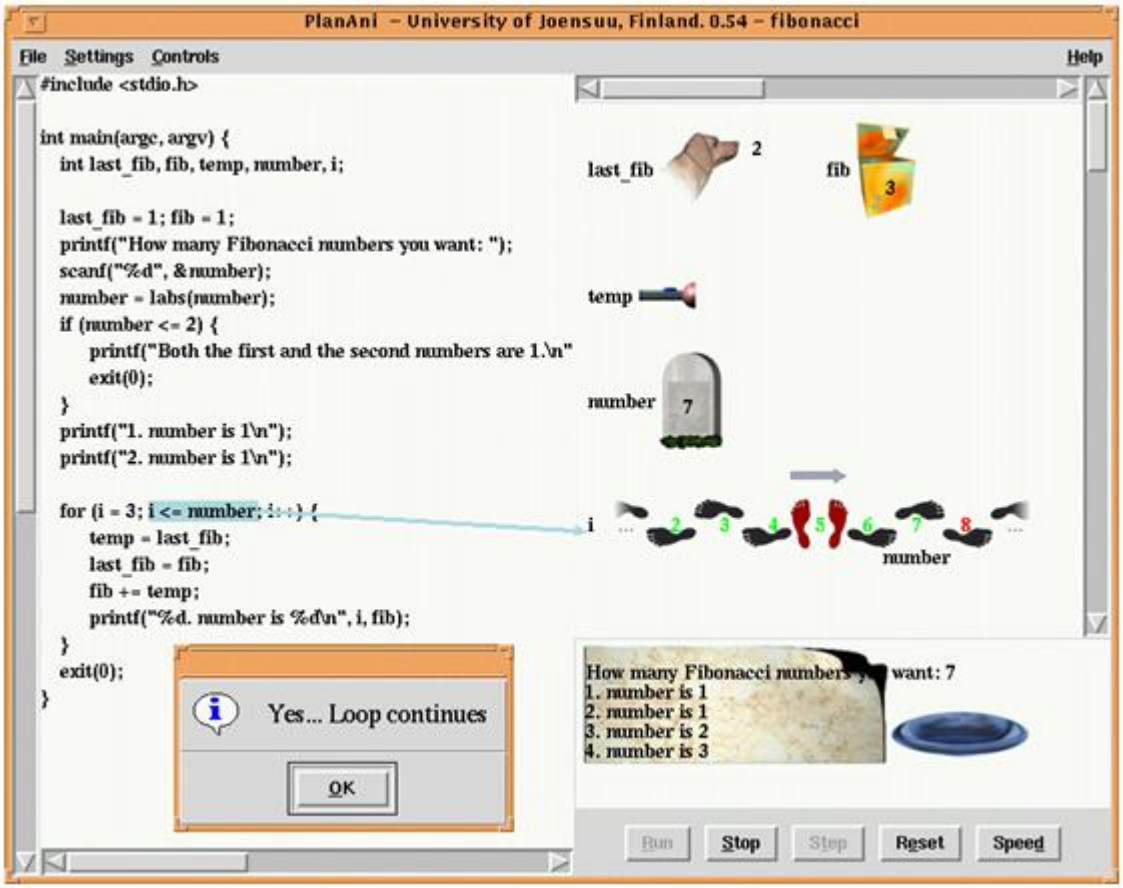


Figura 5.10: Interface do PlanAnim

Dentro da categoria de sistemas de animação e simulação, podemos também referir o ambiente OOP-Anim (Esteves e Mendes, 2004) que recebe código de um programa escrito em Java e mostra como as instruções afectam as classes e os objectos desse programa. Outro ambiente desta categoria é o ambiente SICAS (Gomes, 2000). O SICAS é um ambiente que, enfatizando a apresentação visual, permite ajudar os alunos não apenas a compreender o funcionamento de um algoritmo mas que sobretudo permite conceber, testar, experimentar, alterar e corrigir os algoritmos construídos pelos próprios alunos. Dispondo e interligando os elementos de um fluxograma na área de trabalho os alunos constroem a solução para um determinado problema. O sistema verifica os possíveis erros cometidos pelos alunos e possibilita a animação da solução proposta. Desta forma os alunos podem testar e analisar as soluções por eles construídas ou disponibilizadas pelo professor. A figura 5.11 mostra a interface deste sistema. Neste domínio, destaque-se ainda o sistema H-SICAS (Marcelino et al., 2008) uma nova versão do SICAS desenvolvido para plataformas móveis como PDAs ou *Smart Phones*. Outro ambiente similar ao SICAS é o Raptor12 (Carlise et al., 2005). A figura 5.12 apresenta uma imagem deste ambiente.

[12] Disponível em http://raptor.martincarlisle.com/

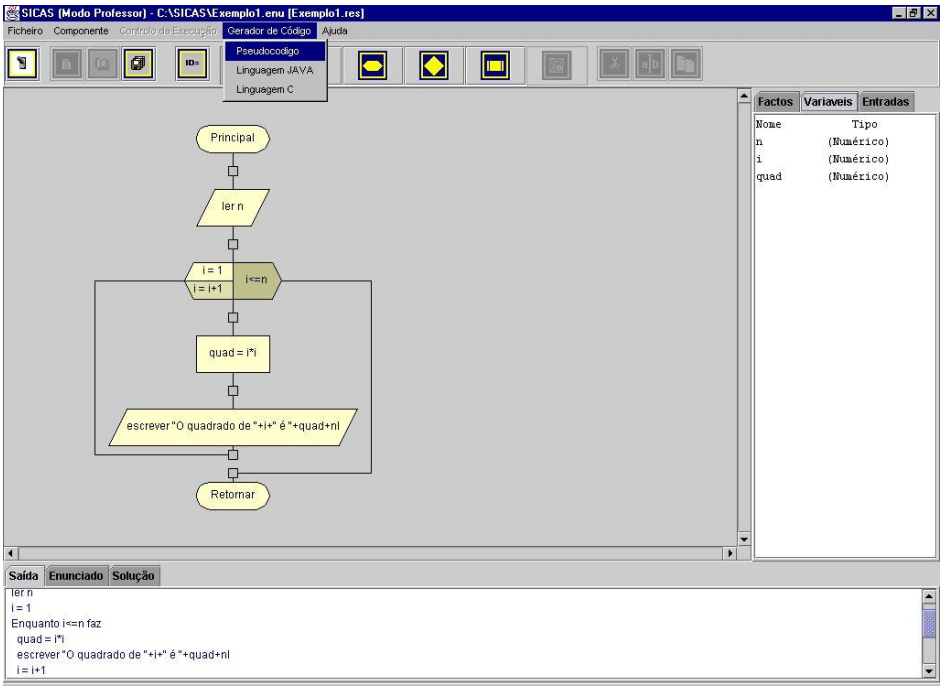


Figura 5.11: Interface do SICAS

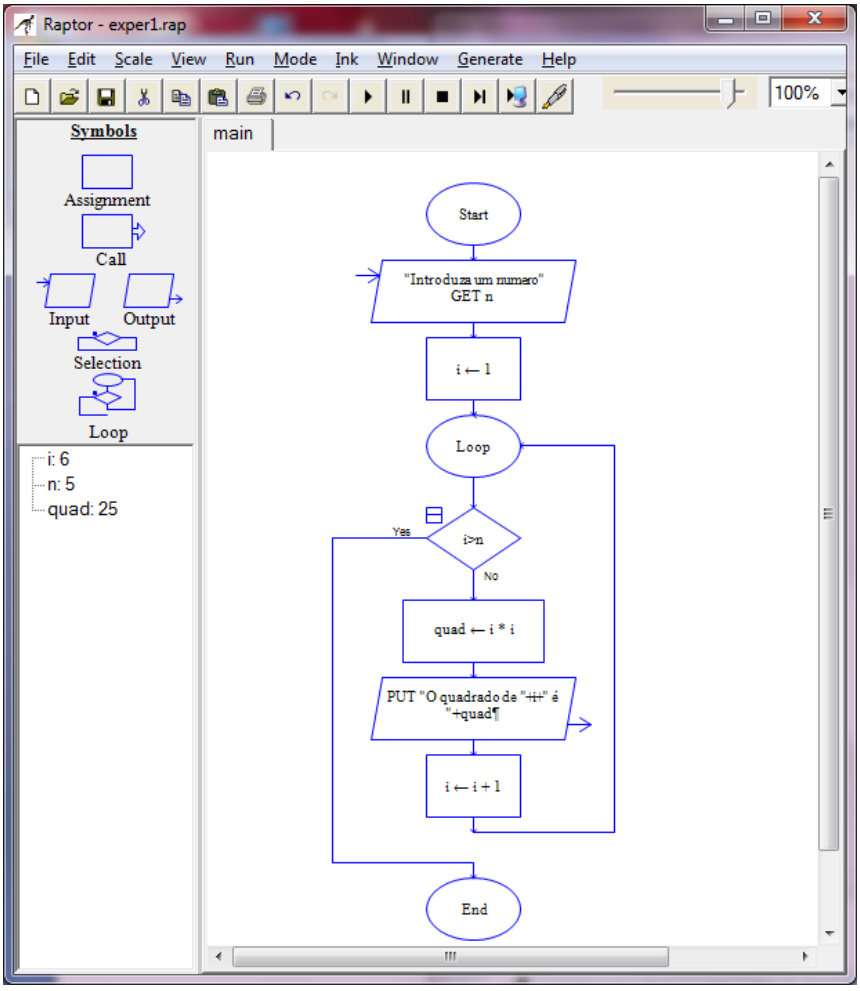


Figura 5.12: Interface do Raptor

5.4 Ambientes Integrados

Outro tipo de ferramentas, são os ambientes integrados, com propósitos mais pedagógicos do que para fins profissionais, resultantes essencialmente de trabalhos académicos. São ambientes menos complexos que os ambientes profissionais, tentando simplificar a aprendizagem inicial de uma linguagem de programação. Por vezes incorporam ainda funcionalidades importantes para programadores pouco experientes.

Um ambiente deste tipo é o Thetis (Freund e Roberts, 1996), um ambiente de desenvolvimento integrado de apoio à criação de programas em linguagem C. Consiste num interpretador de C com uma interface convencional que fornece aos alunos um ambiente simples e intuitivo, permitindo a edição, o *debugging*, a visualização e a execução de programas.

O BlueJ[13] (Kölling et al., 2003) é outro ambiente desta categoria, permitindo o desenvolvimento de programas em Java, especialmente elaborado para níveis iniciais de aprendizagem sobre programação orientada a objectos. Combina edição de código com capacidades de visualização, compilação, *debugging* e execução. É um ambiente que dá uma ênfase especial à visualização e às técnicas de interacção para criar um ambiente interactivo capaz de incentivar a sua exploração. Distingue-se de outros deste tipo por apresentar um diagrama, com a hierarquia das classes do programa, mostrando desse modo a estrutura da aplicação, sendo possível interagir directamente com as classes e objectos, clicando sobre eles. A figura 5.13 apresenta a interface geral deste ambiente e a figura 5.14 a janela de edição correspondente à classe Posição que define uma localização de uma figura no ecrã.

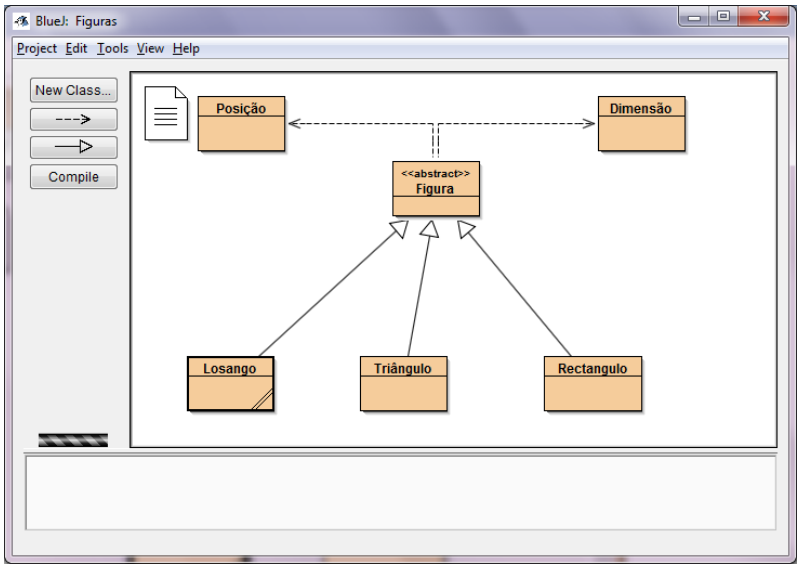


Figura 5.13: Interface principal do BlueJ

[13] Disponível em http://www.bluej.org

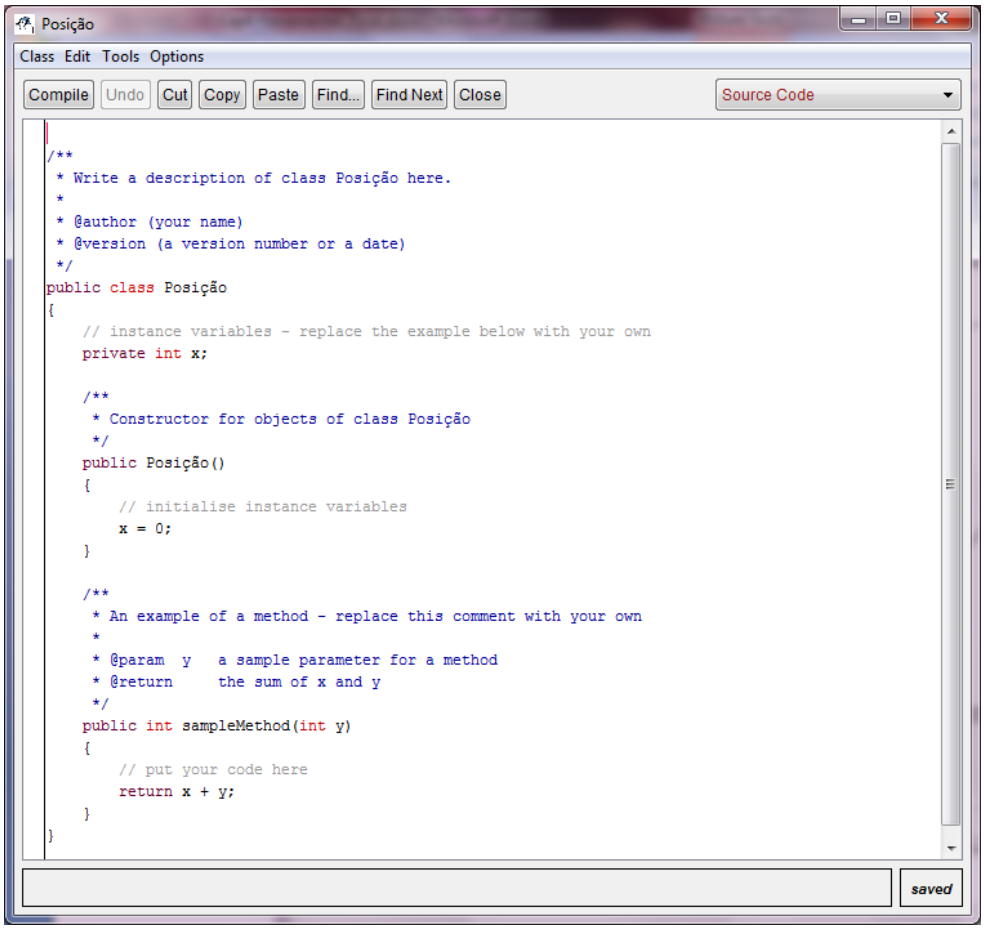


Figura 5.14: Janela de edição do BlueJ

Outro ambiente pedagógico para linguagem Java é o DrJava[14] (Allen et al., 2002). Tal como os outros anteriormente mencionados, o seu principal objectivo é o de focar a atenção dos alunos essencialmente no desenvolvimento de programas, e não na aprendizagem do ambiente. O DrJava tem uma interface simples permitindo ao aluno desenvolver, testar e fazer o *debug* de programas de forma interactiva e incremental. Os autores procuram que o ambiente introduza os mecanismos de escrita de programas de forma agradável, levando os alunos a entender a linguagem por si próprios. Uma característica do DrJava consiste em fornecer uma janela de interacção que permite ao aluno avaliar declarações e expressões em Java, separadamente, facilitando o desenvolvimento incremental do programa. Os alunos podem assim testar o comportamento de determinado pedaço de código. A figura 5.15 apresenta a interface do DrJava e a figura 5.16 mostra a janela de interacções onde é verificado o comportamento de uma estrutura de controlo, um ciclo.

[14] Disponível em http://drjava.org/

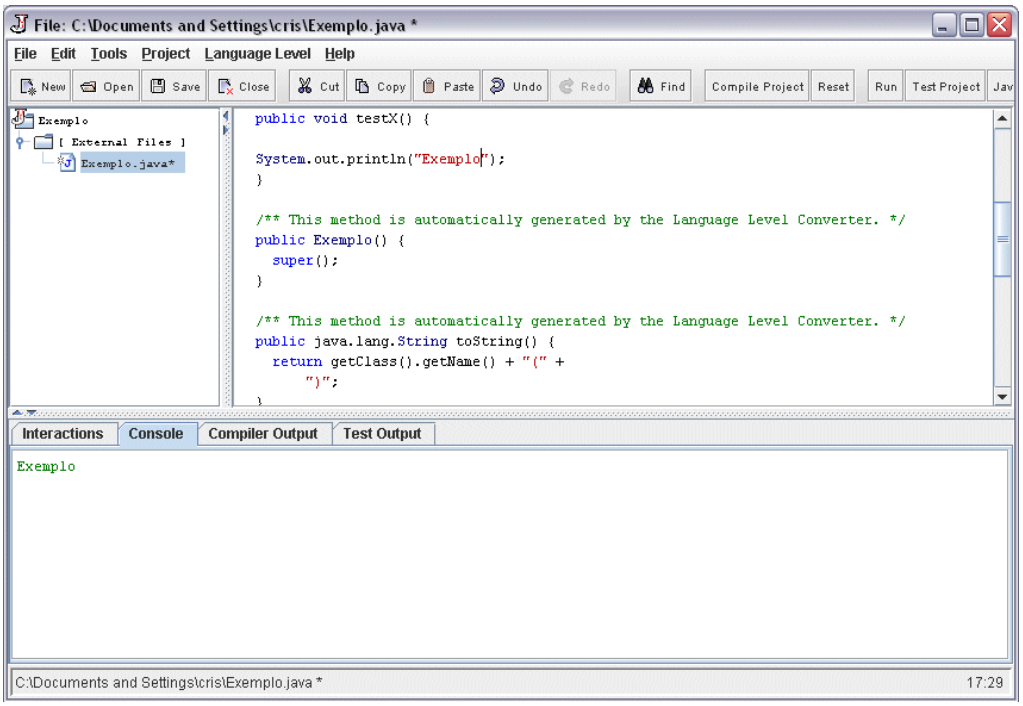


Figura 5.15: Interface do ambiente DrJava

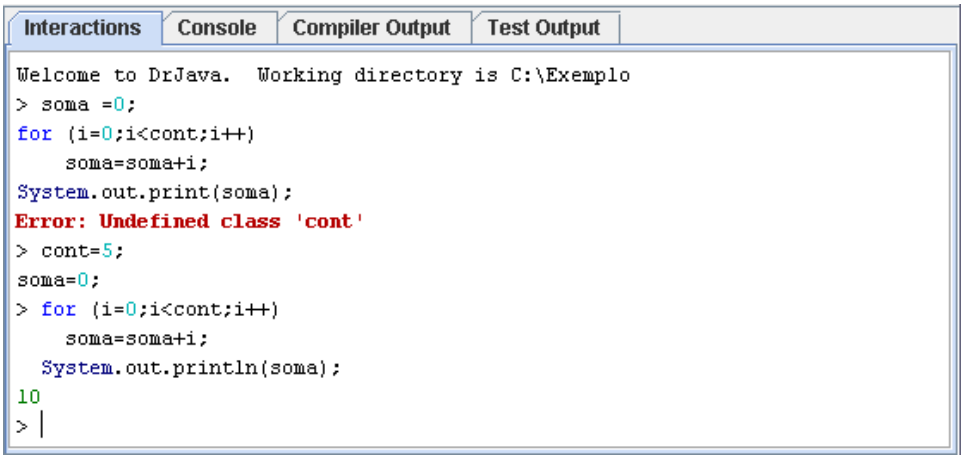


Figura 5.16: Janela de interacções do DrJava

Como exemplos de ambientes deste tipo pode também citar-se o X-Compiler (Evangelidis et al., 2001), o DrScheme [15] (Findler et al., 2002) ou DrPython[16]. Estes dois últimos são ambientes de desenvolvimento para as linguagens Scheme e Python, respectivamente, similares ao Dr. Java. O X-Compiler é um ambiente de desenvolvimento para a linguagem X, um subconjunto do Pascal. Também no contexto da aprendizagem do Pascal, existe o ambiente de desenvolvimento AnimPascal (Satratzemi et al., 2001). Este sistema, para além de auxiliar o aluno na fase de desenvolvimento, verificação, *debug* e execução de programas, permite a animação do programa quando este está a ser executado.

[15] Disponível em http://www.plt-scheme.org/software/drscheme/
[16] Disponível em http://drpython.sourceforge.net

O AnimPascal permite manter o histórico de compilações e registar caminhos de resolução de problemas seguidos pelos alunos, informação esta que poderá ser muito útil para os professores avaliarem as dificuldades e erros dos alunos.

Destaque-se ainda a tendência actual de *plugins* pedagógicos para IDEs profissionais. Refira-se por exemplo, os seguintes desenvolvimentos para o ambiente Eclipse, o CDT Plugin[17], o Gild (Storey et al., 2003), o Penumbra (Mueller e Hosking, 2003), o ProPAT (Barros et al., 2005), o KenyaEclipse (Chatley e Timbul, 2005), o JExercise (Trætteberg e Aalberg, 2006), o Jazz Sangam (Devide et al., 2008) ou o Coala (Monroy, 2010). A título de exemplo refira-se ainda o *plugin* pedagógico para o ambiente NetBeans, o NetBeans IDE BlueJ Plugin[18], resultante da parceria entre o Netbeans e a já abordadada linguagem pedagógica BlueJ. Também baseado no BlueJ, refira-se o Greenfoot (Henriksen e Kölling, 2004), uma ferramenta bastante interessante, constituindo uma plataforma de desenvolvimento que permite uma fácil construção de micromundos animados, implementados em Java. A visualização dos objectos e a interacção com eles são os elementos-chave deste sistema. Este aspecto é especialmente útil para a construção de exercícios de programação que têm um elemento visual, adequado para programadores principiantes. No Greenfoot, a plataforma base ou cenário pode ser escolhido, dentro de um conjunto disponível ou construído pelo programador. Como exemplos típicos de cenários, refiram-se os correspondentes aos populares programas como o "Karel the Robot", "TurtleGraphics" ou "Robocode" ou a tradicionais jogos computacionais como o "Bricks", "Maze" ou "Minesweeper". A figura 5.17 e a figura 5.18 ilustram interfaces deste sistema com diferentes cenários.

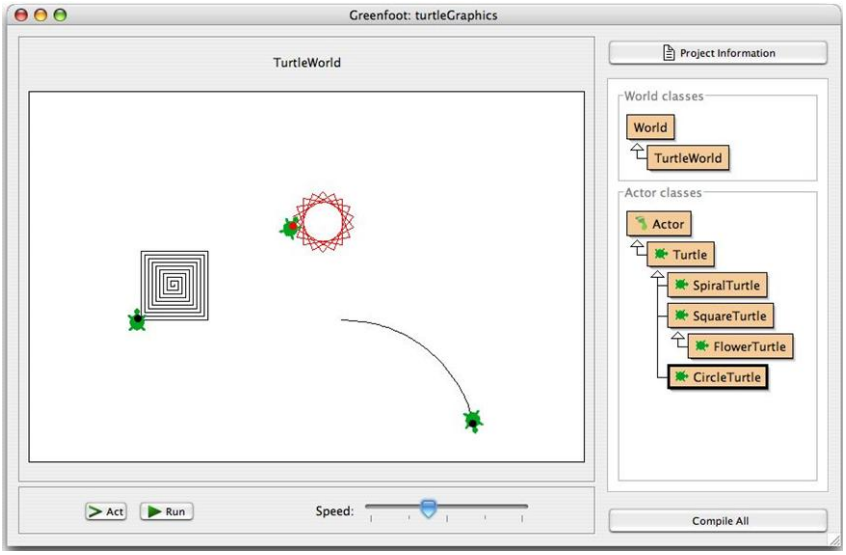


Figura 5.17: Interface do Greenfoot com um cenário do "TurtleGraphics"

[17] Disponível em http://www.eclipse.org/cdt/
[18] Disponível em http://edu.netbeans.org/bluej/

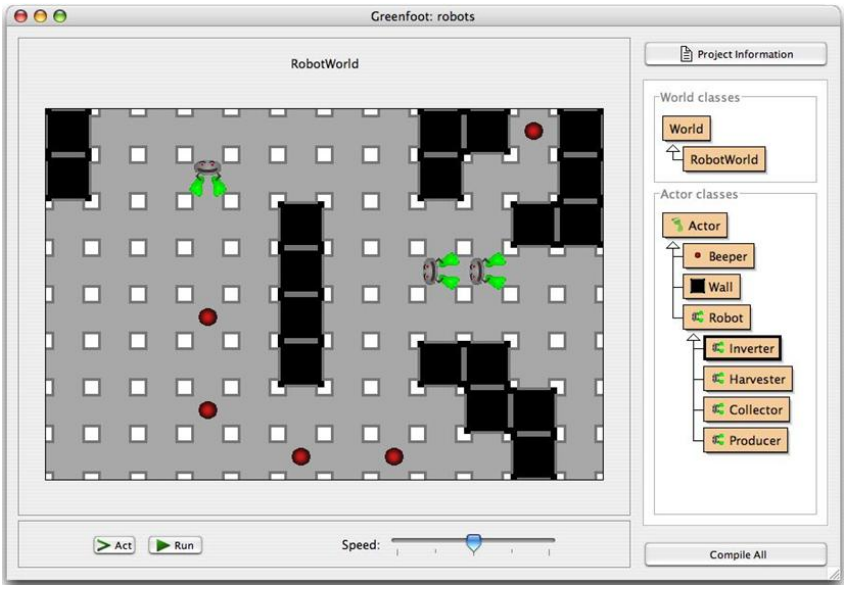


Figura 5.18: Interface do Greenfoot com um cenário do "Karel the robot"

O jGRASP (Cross e Hendrix, 2006) é outro exemplo de um ambiente destinado à aprendizagem da linguagem Java. Também destinado à aprendizagem da linguagem Java, cite-se o jogo educacional e *Open Source* Robocode[19] (Kobayashi et al., 2003), iniciado por Mathew Nelson (originalmente fornecido pela IBM) e actualmente com contribuições de várias pessoas. Este jogo consiste numa batalha de robôs. Cada um deles pode ser programado muito facilmente, porém as estratégias para eliminar outros robôs já poderão ser elaboradas com diferentes níveis de proficiência. Assim, os competidores têm de escrever programas para controlar tanques miniatura que lutam com outros, no campo de batalha, construídos de forma idêntica mas programados diferentemente. Apesar da ideia do jogo poder parecer simples, a estratégia necessária para vencer poderá não sê-lo. Os melhores robôs podem ter centenas de linhas de código dedicadas à estratégia. Alguns dos robôs mais bem sucedidos podem ainda usar técnicas de análise estatística ou de redes neuronais. A figura 5.19 apresenta a interface deste sistema.

[19] Disponível em http://robocode.sourceforge.net/

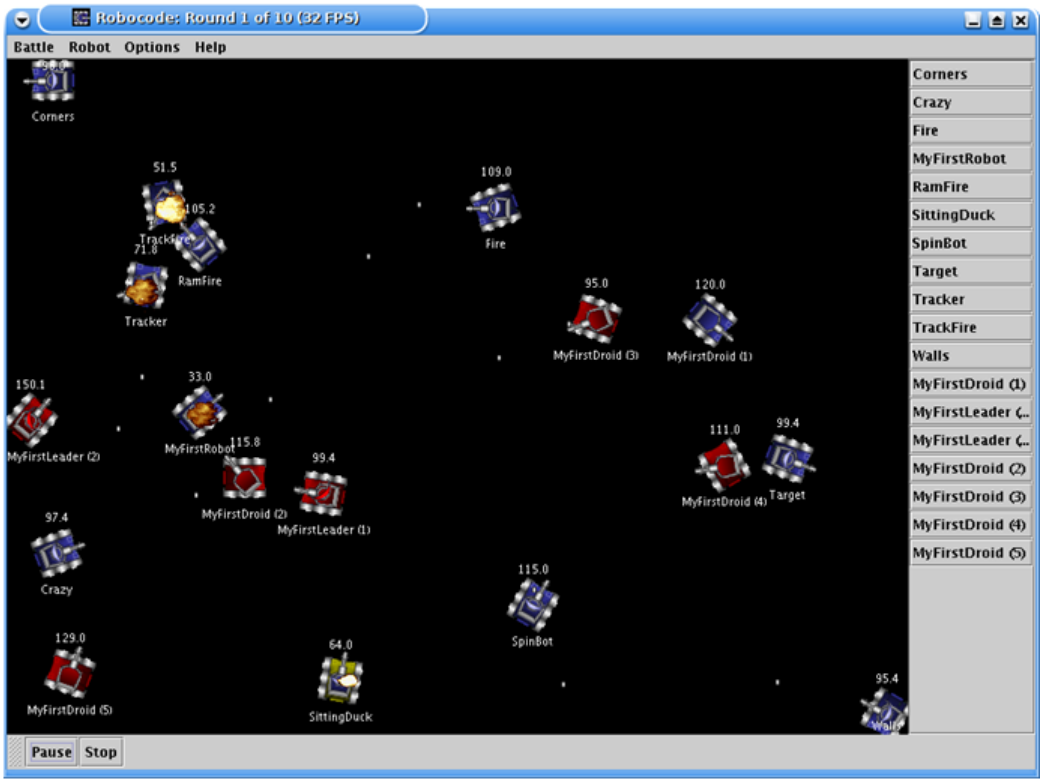


Figura 5.19: Interface do Robocode

5.5 Ambientes Web e de colaboração

Existe também uma grande diversidade de ambientes de aprendizagem, baseados na *world wide web*, *Webworlds*, destinados ao ensino de programação. Nesta categoria é ainda possível encontrar uma grande diversidade de tipos. Muitos destes sistemas limitam-se a apresentar o conteúdo de programação de uma forma estruturada. Existem também sistemas que fornecem actividades adequadamente organizadas e controladas que permitem a tutoria, oferecendo níveis de interactividade adaptados a diferentes necessidades de programação. AnnAnn, AnnAnn.Net (Hooper et al., 2007) e ALEA (Bielikova, 2006) são exemplos desse tipo de sistemas. Outros sistemas incluem diversas outras funcionalidades, entre elas, a possibilidade de fazer a submissão de programas e respectiva avaliação, de que se destacam o WebToTeach (Arnow e Barshay, 1999) e o AulaWeb (García-Beltrán et al., 2005). De notar que o AulaWeb é uma plataforma de e-Learning à qual foram acrescentadas características especiais de que se destacam a possibilidade de submeter tarefas de programação com avaliação automática. Refira-se também a existência de outros sistemas *on-line* que, em geral, pretendem simplificar o processo inicial de aprendizagem de programação, fornecendo editores de código *on-line* e compiladores remotos de programas, de que se destaca o ELP (Truong et al., 2003) ou o WEBLOOP (Cheung, 2006). Baseados nas vantagens oferecidas

pelas plataformas *on-line*, existem também sistemas que introduzem algumas características colaborativas para suportar o ensino e aprendizagem de programação como o Cimel (Wang et al., 2002) ou o JeCo (Moreno et al., 2004).

O nosso grupo de investigação, também já investiu no desenvolvimento de ferramentas colaborativas de suporte à resolução conjunta de problemas de programação. Assim, surgiu a ideia de ligar a ferramenta de simulação SICAS desenvolvida pela equipa da Universidade de Coimbra com a 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), uma nova versão do SICAS com suporte a trabalho colaborativo.

5.6 Tutores inteligentes

Os esforços iniciais para suportar o processo de aprendizagem de programação focaram-se no desenvolvimento de sistemas de tutores inteligentes. Trata-se de sistemas computacionais que recorrem a técnicas de Inteligência Artificial, de forma a promover um ensino personalizado. Um sistema pioneiro deste tipo é o PROUST (Johnson e Soloway, 1985), um sistema de apoio à aprendizagem da linguagem Pascal. É um sistema bastante complexo que propõe determinadas tarefas de programação aos alunos num estágio inicial de aprendizagem. Faz a análise dos programas e apresenta o diagnóstico de erros, mesmo os erros que não sejam sintácticos. Além disso, procura diagnosticar problemas durante o processo de aprendizagem. Num mesmo contexto, destaca-se também o LISP Tutor (Anderson e Reiser, 1985), sistema muito utilizado no ensino de programação em Lisp, o qual monitoriza constantemente as entradas do aluno de forma a detectar os erros e dificuldades na resolução de determinado problema, guiando-o pelo caminho correcto. Também neste campo, podemos encontrar o INTELLITUTOR (Ueno, 1989) que para além de fazer o *debug* de programas elaborados em Pascal ou C tenta guiar o aluno na resolução de um problema. Também o sistema Bridge (Bonar et al., 1988) foi construído de forma a ajudar os alunos a edificar a descrição da solução de um problema em linguagem natural, resultando num programa. Estes sistemas auxiliam fundamentalmente na fase da

implementação, no entanto existem outros sistemas que também prestam ajuda na fase de análise e planeamento de um programa, como por exemplo o *The Programmer's Apprendice* (Rich e Waters, 1988) e o *Coached Program Planning* (Lane e VanLehn, 2003). Outros exemplos de sistemas de tutores inteligentes são o DISCOVER (Ramadhan, 1992) que auxilia o ensino de programação através de uma linguagem própria, semelhante ao pseudocódigo, o C-Tutor (Song et al., 1997), um sistema tutor inteligente para ensino da linguagem C, o sistema ELM-ART (Weber e Brusilovsky, 2001) para o ensino da linguagem de programação LISP, através da *world wide web*. Este foi desenvolvido no sistema ELM-PE (Weber, 1996) um ambiente de aprendizagem que suporta a programação baseada em exemplos, análise das soluções do problema, teste das soluções e respectivo *debug*.

Ainda que haja alguns sistemas conceptualmente bastante interessantes, a verdade é que muitos dos sistemas tutores inteligentes propostos na literatura são bastante complexos, não havendo evidência da sua utilização frequente em contextos reais de aprendizagem. Além disso, este tipo de sistemas apresenta uma limitação, na nossa óptica, crucial, o facto de apenas poderem dar apoio a problemas que se encontram na sua base de conhecimento.

Uma evolução recente é apresentada em Kumar (2005a, 2005b). Trata-se de um sistema tutor inteligente relativamente recente que automaticamente gera problemas, classificações às respostas dos alunos e *feedback* às suas respostas em função de um conjunto de tópicos seleccionados. O autor está actualmente a desenvolver uma série de tutores automatizados para cursos introdutórios de programação. Neste sistema, os tutores são adaptáveis, isto é, geram os problemas adaptados às necessidades de aprendizagem do aluno.

5.7 Sistemas de avaliação automática

De certa forma relacionado com os sistemas referidos anteriormente, surgiram os sistemas de avaliação automática de programas. Há actualmente diversos sistemas para avaliação de diferentes aspectos relativos à programação realizada pelos alunos. Existem sistemas que testam as respostas dos alunos face a um conjunto de dados de entrada, de que se destacam, Assyst (Jackson e Usher, 1997), BOSS (Luck e Joy, 1999), Ceilidh (Benford et al., 1993; Foxley, 1999), RoboProf (Daly, 1999), Ceilidh CourseMarker (Higgins et al., 2003), Online Judge (Cheang et al., 2003), HoGG (Morris, 2003) ou o Mooshak[20]. Outros sistemas avaliam automaticamente a funcionalidade de pequenas entidades, como funções ou métodos, em vez de programas completos. Refira-se por exemplo, Cecchi (Bettini et al., 2004) ou Quiver (Ellsworth et al., 2004). Outros testam programas com interfaces gráficas,

[20] Disponível em http://mooshak.dcc.fc.up.pt/

destacando-se o JEWL (English, 2004). Outros ainda testam programas relativamente ao estilo da codificação utilizada, como por exemplo, o PASS (PASS) ou o Style++ (Ala-Mutka e Järvinen, 2004). Quizpack (Brusilovsky e Pathak, 2002; Sosnovsky et al., 2003; Brusilovsky e Sosnovsky, 2005) é um exemplo de um sistema que avalia a compreensão de programas. Edwards é um exemplo de um sistema inclusivo que avalia a geração de programas baseado numa aproximação de programação *test-driven* (Edwards 2003a, 2003b). O sistema fornece *feedback* concreto sobre quais as partes do programa que exigem (mais) testes e sobre o estilo da codificação. Os sistemas TRAKLA (Hyvönen e Malmi, 1993) e TRAKLA2 (Korhonen et al., 2003), são também exemplos de sistemas populares, projectados especificamente para avaliar exercícios de simulação de algoritmos, permitindo que os estudantes submetam novamente as suas soluções após terem recebido *feedback* do sistema. O projecto de dois cursos que aplicaram o sistema TRAKLA2 é descrito em Malmi et al. (2005) e Malmi e Korhonen (2008).

Apesar de existirem alguns sistemas para avaliação automatizada, apenas recentemente, começaram a surgir alguns trabalhos referentes a apreciações desses sistemas (Korhonen et al., 2002; Kumar, 2005b; Malmi et al., 2005; Karavirta et al., 2006; Traynor et al., 2006), sendo também necessários estudos que façam a comparação entre diferentes sistemas. A avaliação automatizada é coberta por duas revisões recentes. Alá-Mutka fornece um levantamento de abordagens de avaliação automatizadas para tarefas de programação (Ala-Mutka, 2005) e o JERIC (*Journal of Educational Resources in Computing*) teve recentemente uma edição especial sobre avaliação automatizada de tarefas de programação (Brusilovsky e Higgins, 2005). Desta edição especial, destaca-se em particular o artigo de Douce e colegas sobre avaliação automática de programação baseada em testes (Douce et al., 2005).

Destaque-se ainda os Problets[21], como outro popular e recente exemplo deste tipo de sistemas. Os *problets* são assistentes para a resolução de problemas de programação. Cada *problet* gera problemas, avalia a resposta do aluno e explica a solução correcta. O sistema gera cada problema como uma instância de um *template* de problemas. Como resultado, dois problemas gerados nunca são iguais. Cada aluno, de cada vez que usa o sistema terá um conjunto de problemas diferenciado, para impedir o plágio e maximizar a aprendizagem. Cada *problet* consiste num repositório de cerca de 200 *templates* de problemas, gerando problemas de forma adequada ao conhecimento do aluno e sempre sobre os conceitos que o aluno ainda não domina. Segundo os seus autores, esta característica minimiza o tempo de aprendizagem e melhora o envolvimento e interesse do aluno.

O plágio é também um tema importante no ensino de ciências da computação como exposto por um grupo de trabalho da "ITiCSE 2002" (Dick et al., 2003). Diversos sistemas da

[21] Disponível em http://www.problets.org/

avaliação incorporam a detecção de plágio, nomeadamente, BOSS (Luck e Joy, 1999), CourseMarker (Higgins et al., 2003) ou Online Judge (Cheang et al., 2003). Lancaster e Culwin (2004) fornecem uma comparação de mecanismos para detecção de plágio em código fonte. Daly e Horgan (2005) apresentam um sistema baseado em marcas de água, permitindo distinguir o fornecedor e o receptor, no caso de plágio. Verco e Wise (1996) compararam diferentes ferramentas automáticas utilizando diferentes métodos e abordagens para detectar cópias de programas, nomeadamente em sistemas como o MOSS (Aiken, 1994) e o JPlag (Malpohl, 1996).

5.8 Conclusões

Constata-se que os alunos enfrentam uma multiplicidade de dificuldades aquando do contacto com um primeiro ambiente de programação. Na realidade, a maioria destes ambientes são concebidos recheados de funcionalidades, mais orientados para fins profissionais do que com finalidades pedagógicas. No entanto, existe uma multiplicidade de ferramentas que foram desenvolvidas ao longo dos tempos com o intuito de ajudar os alunos a ultrapassar as dificuldades iniciais associadas ao desenvolvimento dos primeiros programas. Cremos que a maioria delas tem os seus próprios méritos, no entanto, pensamos que algumas são mais apropriadas, por adoptarem uma abordagem construtivista da aprendizagem. Desta forma, apresentámos e descrevemos, embora de forma abreviada, aquelas que, na nossa óptica poderão dar algum contributo no sentido de promover a aquisição de competências necessárias para uma programação inicial bem sucedida. Em geral, consideramos de crucial interesse a utilização de micromundos de aprendizagem, bem como a utilização de ferramentas que permitam que os alunos construam as suas próprias soluções e posteriormente permitam analisar o comportamento das soluções construídas, de preferência recorrendo a múltiplas representações.

Cap. 6

Estudos realizados

"Nem tudo o que pode ser contado conta, e nem tudo o que conta pode ser contado."

Albert Einstein

6.1 Introdução

Como já referido em capítulos anteriores, pensamos que os factores mais importantes com influência na aprendizagem introdutória de programação, estão relacionados com três aspectos, nomeadamente os métodos de ensino, os métodos de aprendizagem e o conhecimento prévio dos alunos. Estas causas já foram consubstanciadas em factores de investigação que passamos a reproduzir.

F1 – Muitos alunos com dificuldades de aprendizagem de programação não possuem as competências necessárias exigidas por uma disciplina introdutória de programação.

F1.1 – Muitos alunos com dificuldades de aprendizagem de programação apresentam muitas dificuldades em resolver problemas.

F1.2 – Muitos alunos com dificuldades de aprendizagem de programação apresentam défices de conhecimentos matemáticos e lógicos.

F2 – As condições e métodos de ensino habitualmente utilizados não são os mais adequados para o ensino de programação.

F2.1 – A Engenharia Informática/Ciências da Computação atrai alunos com determinado perfil de estilos de aprendizagem.

F2.2 – Existe correlação entre os estilos de aprendizagem dos alunos e os seus resultados à primeira disciplina de programação.

F2.3 – As tarefas habitualmente propostas pelos professores apresentam níveis de dificuldades desajustados ao nível cognitivo do aluno.

F3 – Os métodos de estudo utilizados pelos alunos não são os mais adequados para a aprendizagem em geral e da programação em particular.

F4 – As percepções pessoais dos alunos são em geral muito baixas, insuficientes para enfrentar as exigências das disciplinas de programação.

Neste capítulo tentaremos dar resposta a estes aspectos através de um conjunto de estudos, de seguida apresentados.

6.1.1 ILS

Em muitos dos nossos estudos utilizámos o *Index of Learning Styles* (ILS), já descrito na secção 3.4, para caracterizar os estilos de aprendizagem dos alunos envolvidos. Esta secção tem como objectivo justificar a razão da escolha desse instrumento. James e Blank (1993) referem um conjunto de factores a considerar aquando da selecção de determinado instrumento para determinar os estilos de aprendizagem de um indivíduo. Consideram que os factores devem ser organizados em três áreas, nomeadamente: a base conceptual, os dados investigados e considerações práticas. A base conceptual de um instrumento é a base teórica sobre a qual o instrumento foi desenvolvido. Por outro lado, quando se analisam os dados investigados é importante ter em consideração a validade, a fiabilidade e normas que incluam padrões de comparação (por idade, género, nível educacional entre outros parâmetros relevantes). Segundo estes autores, a terceira área de avaliação de um instrumento, prende-se com considerações práticas.

Porém, a nossa opção por este modelo surgiu predominantemente pelo facto de Felder, o criador do instrumento e modelo referido, pertencer à área das engenharias e direccionar as suas pesquisas de estilos de aprendizagem para essa área. Dado o público-alvo a estudar, pareceu-nos fundamental utilizar um instrumento desenvolvido atendendo às especificidades da população pretendida. Outro aspecto procurado foi o da averiguação da validade do instrumento utilizado. Após a publicação do ILS, vários investigadores realizaram estudos com a finalidade de calcular os seus coeficientes de validade, nomeadamente, Livesay et al. (2002), Seery et al. (2003), Zywno (2003), Litzinger et al. (2007), entre outros. No que se refere ao coeficiente de confiabilidade, os testes foram administrados com intervalos de quatro semanas e de sete a oito meses. As investigações em ambos os

intervalos chegaram a resultados satisfatórios. No que respeita à consistência interna, calculada pelo coeficiente de alpha de Cronbach, os valores obtidos em todas as quatro dimensões foram superiores a 0,5. Face a estes resultados, Felder e Spurlin (2005) salientam que, quando usado adequadamente, o ILS tem contribuído para auxiliar o professor a adequar as aulas às diferentes formas de aprendizagem, além de ter sido útil para ajudar os alunos no seu desenvolvimento pessoal e académico. Daí termos considerado que o ILS estaria validado e apto para ser utilizado. Não obstante, concordamos com McKeachie (1995) ao salientar que, independentemente da validade do instrumento usado, o importante é que o professor reconheça que os seus alunos são diferentes e considerem essas diferenças no ensino.

Adicionalmente e deveras por razões práticas, este revelou-se o melhor modelo. A facilidade de administração, através de um inquérito *on-line*, a geração automática dos resultados, de forma extremamente fácil de interpretar (após se ter respondido a todas as questões, bastava premir o botão "*Submit*" que os dados eram automaticamente gerados e enviados para o endereço do respondente) revelaram-se factores determinantes. A versão *online* encontra-se em inglês. No entanto, para evitar interpretações erradas por parte dos respondentes, nas experiências relatadas neste trabalho, foi utilizada a versão portuguesa traduzida por Marcius Giorgetti (Anexo G), após a um pedido nosso ao autor.

6.2 Estudo A

Este estudo teve várias finalidades (Gomes et al., 2006). Um dos objectivos consistiu em estudar os factores F1.1 e F1.2, analisando as competências de resolução de problemas e conhecimentos matemáticos de alunos com dificuldades a programação. Pretendeu-se também verificar se os alunos com dificuldades a programação, quando submetidos a um reforço de resolução de problemas matemáticos e lógicos melhoravam as suas competências de programação.

Simultaneamente tentou-se aprofundar o conhecimento teórico relativo aos estilos de aprendizagem e constatar se as estratégias utilizadas pelos alunos ao resolverem problemas condiziam com as características teóricas enunciadas pelo modelo utilizado e assim ganhar sensibilidade ao tema. Por outro lado, pretendeu-se averiguar se existiam padrões de estilos de aprendizagem entre os alunos com dificuldades a programação e neste sentido estudar em parte o factor F2.1.

6.2.1 Contexto

O estudo foi realizado durante o 2º semestre de 2006/2007 tendo participado 29 alunos do curso de Licenciatura em Engenharia Informática e de Sistemas, do Instituto Superior de Engenharia de Coimbra (LEIS-ISEC). Os alunos foram seleccionados no âmbito da disciplina de Matemática Discreta. Por pensarmos que os alunos com dificuldades a programação apresentam défices em termos de conhecimentos matemáticos, bem como limitações ao nível da abstracção e do raciocínio lógico, que de forma implícita ou explícita está inerente a muitos dos problemas de programação, recorremos a uma turma constituída por alunos que tinham reprovado à 1ª disciplina de programação, no semestre anterior.

6.2.2 Instrumentos

Cada aluno foi submetido, numa sessão inicial, ao teste *Index of Learning Styles* (Anexo G), respondido *on-line*. Os alunos foram ainda submetidos a um teste de diagnóstico de resolução de problemas (Anexo A.1), um teste de diagnóstico de programação (Anexo A.2), testes semanais sobre resolução de problemas de base matemática (Anexos A.3, A.4, A.5 e A.6), bem como a um teste final de programação (Anexo A.7). A ideia subjacente aos testes de diagnóstico e final de programação era a de verificar os conhecimentos iniciais dos alunos, bem como a sua progressão ao longo das sessões, respectivamente. O objectivo do teste de diagnóstico sobre resolução de problemas era o de aferir alguns aspectos referentes à capacidade de resolução de problemas lógicos e outros com alguma base matemática.

Os instrumentos de pesquisa foram aplicados em sala de aula, em horários previamente combinados entre a investigadora e os alunos participantes e com a contribuição da docente responsável pela disciplina de Matemática Discreta. Inicialmente os alunos receberam uma breve explicação sobre a pesquisa e os seus objectivos após o que receberam os instrumentos e respectivas instruções.

6.2.3 Metodologia

A metodologia genérica utilizada neste estudo foi o estudo de caso. Pretendia-se esclarecer se havia uma relação entre as variáveis capacidade de resolução de problemas e competências matemáticas e as dificuldades a programação, descrevendo estes aspectos numa amostra constituída por alunos com dificuldades a programação. Neste caso, focámo-nos num grupo e não fizemos comparações entre este grupo com outros, mas tentámos estudar o fenómeno no seu contexto real, antes de efectuar testes mais rigorosos. Em simultâneo, podemos dizer que este estudo de caso, teve uma natureza exploratória, procurando confirmar se os conceitos teóricos referentes aos estilos de aprendizagem se

verificavam na prática. Consideramos que estas provas de conceito seriam de grande utilidade para ganhar sensibilidade a uma realidade complexa e praticamente desconhecida na altura. A aquisição desse conhecimento permitiu identificar as variáveis mais relevantes do fenómeno em estudo e recolher informação para descrever o que se passava.

Nesta experiência realizaram-se sessões semanais de actividades de resolução de problemas. Cada sessão tinha a duração de uma hora, durante a qual era distribuído a cada aluno um teste com diferentes perguntas, às quais os alunos tinham de responder de forma escrita. As perguntas incidiam sobre aspectos de resolução de problemas. Nos problemas apresentados a tónica incidia sobre lógica e desafios matemáticos, incluindo principalmente, de forma implícita ou explícita, conceitos matemáticos considerados importantes para obter um bom desempenho a disciplinas de programação.

À excepção da primeira, cada sessão era normalmente iniciada com o esclarecimento de alguns aspectos que a investigadora tinha detectado como problemáticos na sessão anterior. Este tipo de esclarecimento ocupava normalmente muito pouco tempo da sessão, pois a maioria dos alunos revelava um interesse elevado pelo tipo de exercícios, consultando os professores ou discutindo com os colegas, activamente, as soluções alguns dias antes da sessão seguinte. Esta motivação pôde também ser constatada pelo facto de as sessões ocorrerem às oito horas da manhã de sexta-feira, o único horário compatível entre alunos e investigadora e, durante todo o semestre em que ocorreram, não ter havido qualquer desistência.

Apesar dos testes de diagnóstico fornecerem alguns indicadores sobre o tipo de dificuldades dos alunos, fizemos ainda várias pesquisas no sentido de orientar a escolha dos exercícios a incluir nas experiências. Decidiu-se seguir o conjunto de normas para o currículo e avaliação em matemática escolar[22]. Nesse documento são definidas normas que indicam competências que a matemática deve promover, para diferentes escalões de ensino. Os escalões são divididos em três grupos, o referente ao ensino básico, o compreendido entre o 5º e o 8º anos de escolaridade e o que engloba os 9º, 10º, 11º e 12º anos de escolaridade. Após a análise das propostas de exercícios e competências definidas em cada um dos níveis, decidiu-se concentrar a experiência tendo como base algumas normas relativas aos graus compreendidos entre o 5º e o 8º ano de escolaridade (Anexo A.8). Esta decisão foi tomada, não por se pensar que os alunos não conseguiriam resolver problemas de anos superiores, mas por se considerar que é nesse tipo de problemas que os alunos têm mais dificuldades e os que mais influência têm sobre a capacidade de programar.

[22] *Tradução Portuguesa dos Standards do National Council of Teachers of Mathematics - Associação de Professores de Matemática*. (Out. de 1999).

6.2.4 Análise de resultados

Tendo como base as respostas dos alunos aos diferentes instrumentos utilizados, bem como as observações da investigadora, foi possível constatar que os alunos envolvidos apresentavam dificuldades de diversa ordem:

- Não dominavam conceitos matemáticos básicos referentes à Teoria de Números. Uma vez que muitos dos exercícios de programação envolvem conceitos deste tipo, este poderá ser um dos factores a contribuir para o insucesso a programação. Esta dificuldade compromete a realização de uma panóplia de exercícios de programação que implicam que os alunos apliquem implícita ou explicitamente conceitos relativos à teoria dos números (por exemplo, números primos, divisores, múltiplos, entre outros).

- Tinham dificuldade de cálculo. Pensa-se que os alunos muitas vezes não conseguem programar porque o tipo de exercícios que lhes é pedido para resolver implica cálculo. É necessário que os alunos desenvolvam procedimentos para a determinação de certas quantidades, com o objectivo de resolver problemas. Constatou-se, por vezes, que os alunos conseguiram resolver um problema assumindo quantidades concretas, mas não conseguiam derivar expressões para o cálculo dessas quantidades partindo de um valor geral e abstracto.

- Eram incapazes de traduzir um problema numa fórmula matemática. Quando o enunciado envolvia quantidades facilmente calculáveis mentalmente ou "contando pelos dedos" a maioria dos alunos conseguia resolvê-los correctamente. Porém, constatou-se frequentemente que, quando lhes era pedido posteriormente para obter a solução do mesmo problema de uma forma genérica, através de uma fórmula, a maioria já não o conseguia fazer. Assim, a dificuldade de implementação de alguns problemas de programação poderá estar relacionada com o facto de os alunos não conseguirem passar da descrição textual de um problema para a fórmula matemática que o resolve.

- Desconheciam figuras geométricas. Verificou-se que os alunos desconheciam ou pelo menos não tinham bem clara a definição de conceitos relativos a figuras geométricas. Este poderá também ser um dos factores eventualmente responsável pelo insucesso na resolução de muitos problemas de programação, traduzindo-se na incapacidade de os alunos identificarem, classificarem, descreverem e compararem figuras geométricas.

- Tinham dificuldades de interpretação do enunciado. Pensa-se que muitas vezes os alunos não conseguiam resolver um problema por não entenderem completamente o enunciado. Apesar de os exercícios terem sido escritos de uma forma que se entendeu clara e esclarecedora, os alunos continuavam a apresentar frequentemente dúvidas de interpretação. Durante as sessões, os alunos constantemente perguntavam o que era pedido. Por vezes, os professores envolvidos pediam aos alunos que lessem o enunciado calmamente, em voz alta, verificando que eles não sabiam ler, pois faziam pausas ou omitiam-nas em sítios errados, deturpando completamente o sentido do texto. Posteriormente, quando o professor lhes lia o enunciado, diziam que já tinham percebido, mesmo sem ter sido dada qualquer explicação adicional. Outras vezes os alunos referiam que não sabiam por onde iniciar o processo de resolução, por não perceberem claramente o que era pedido. Porém, quando lhes era apresentado um problema equivalente mas dividido em várias alíneas correspondentes às etapas necessárias para o resolver, muitos deles, já o conseguiam fazer. Este aspecto levantou outro problema, pois verificou-se que quando os problemas eram apresentados desta forma, os alunos nem sempre relacionavam as diversas alíneas entre elas e poucos percebiam que cada uma delas contribuía para a resolução como um todo.

- Eram incapazes de definir critérios de comparação para tomar decisões. Mesmo quando resolviam os problemas correctamente, frequentemente não conseguiam explicar o processo de tomada de decisão. Durante as experiências, frequentemente se verificou que faltavam aos alunos detalhes importantes para a resolução de um problema e que as generalizações que conseguiam fazer eram insuficientes.

- Eram incapazes de se orientar no plano cartesiano, tinham falta de conceitos trigonométricos ou eram incapazes de os aplicar à resolução de exercícios. Pensa-se que muitas vezes os alunos não sabiam resolver problemas de programação porque não se apercebiam da aplicação dos modelos geométricos necessários à resolução desses problemas. Constatou-se que muitos alunos não conseguiam posicionar circunferências em determinadas coordenadas e outros desconheciam conceitos trigonométricos fundamentais como seno e co-seno. Porém, muitos outros possuíam esses conceitos mas revelaram-se incapazes de os aplicar ao problema proposto.

- Tinham fracos níveis de abstracção. Vários problemas apontados anteriormente também podem estar relacionados com a grande dificuldade de abstracção

evidenciada pelos alunos. Nas experiências realizadas, constatou-se que os alunos quando submetidos a problemas da mesma natureza apresentavam desempenhos diferentes consoante as quantidades envolvidas fossem concretas ou abstractas.

- Tinham falta de treino lógico e de raciocínio. Ao longo das experiências, foram lançados vários desafios lógicos e charadas que envolviam a exclusão de partes, lógica de proposições, descoberta de ciladas ou raciocínios errados, entre outros problemas de raciocínio, apenas com a finalidade de treinar o raciocínio lógico, mesmo que não tivesse aplicação directa na programação. Constatou-se que inicialmente os alunos apresentaram grandes dificuldades. Contudo observou-se que este tipo de dificuldades foi significativamente atenuado à medida que foi exercitado ao longo das sessões. Consideramos ser importante para a capacidade de programar que os alunos consigam aplicar o raciocínio indutivo e dedutivo, bem como validar o seu próprio pensamento e que este seja intensivamente treinado.

Cremos que os resultados anteriormente apresentados, suportam as ideias de partida relativamente aos factores F1.1 – Muitos alunos com dificuldades de aprendizagem de programação apresentam muitas dificuldades em resolver problemas e F1.2 – Muitos alunos com dificuldades de aprendizagem de programação apresentam défices de conhecimentos matemáticos e lógicos.

Relativamente ao estudo dos estilos de aprendizagem dos alunos, factor F2.1, os resultados da aplicação do teste ILS, encontram-se na figura 6.1. Embora a amostra fosse muito pequena e específica (incluía apenas alunos voluntários que tinham reprovado à primeira disciplina de programação) os resultados são consonantes com o que se encontra na literatura relativamente a alunos de engenharia informática. Desta forma constatou-se que a maioria dos alunos da amostra era sensorial, visual, activa e sequencial.

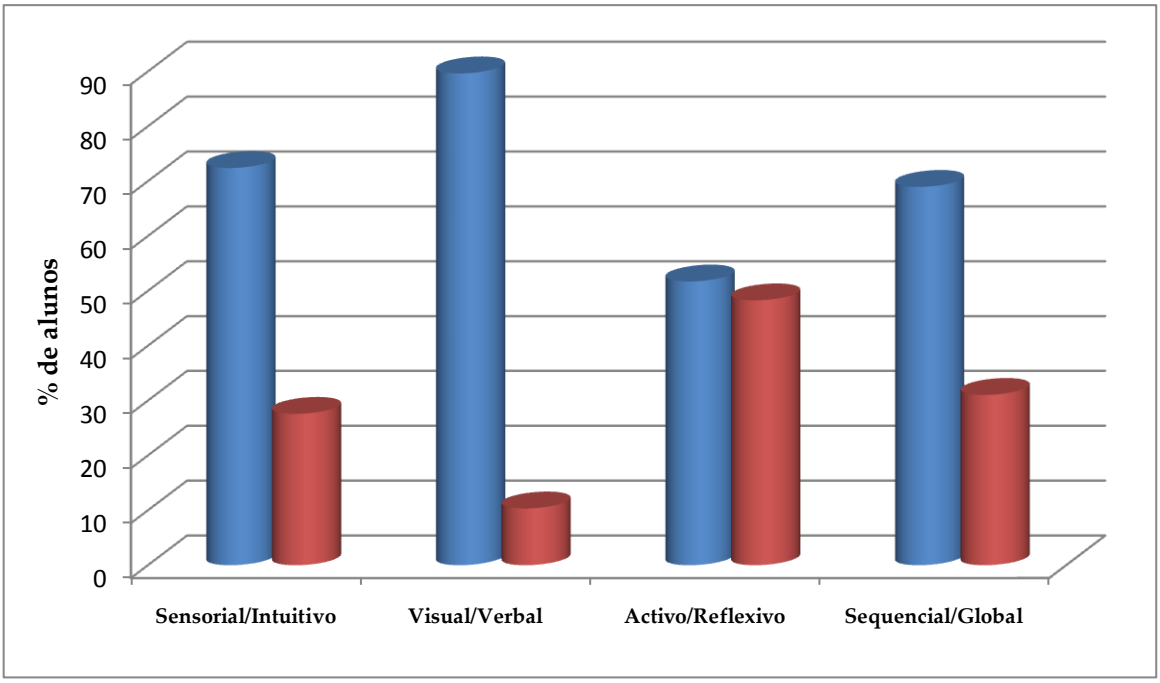


Figura 6.1: Estilos de aprendizagem

Apresentamos de seguida os resultados relativos a outro dos propósitos deste estudo, o qual objectivava comparar as estratégias de resolução de problemas dos alunos com os seus estilos de aprendizagem predominantes. A maioria significativa dos alunos testados apresentava um estilo de aprendizagem marcadamente visual em detrimento de verbal. Analisando as resoluções dos alunos visuais verificou-se que neles predominavam as seguintes características:

- Obtinham melhores desempenhos nos exercícios cujo enunciado envolvia figuras do que naqueles traduzidos apenas através de descrições textuais.

- Tinham mais facilidade em estruturar as suas respostas de forma esquemática, geralmente apresentavam a solução de forma gráfica/ilustrada. Verificou-se inclusivamente que quando confrontados com pedidos de descrição textual e esquemática para o mesmo tipo de problema, pareciam revelar conhecimentos distintos.

- Davam explicações muito sucintas e por vezes até insuficientes, talvez por conseguirem imaginar melhor a solução e não se aperceberem da necessidade de detalhar as explicações escritas. Isto porque, quando eram induzidos a detalhar a resposta muitas vezes sabiam fazê-lo. Em contrapartida os verbais conseguiam melhores descrições textuais das suas resoluções.

Assim, a descrição textual dada pelos visuais encontrava-se geralmente bastante incompleta, porém, a representação gráfica era, em geral muito clara e bem sucedida. Por vezes, também se verificou que a dificuldade de expressão verbal apresentada pelos visuais era tal que, frequentemente, deturpavam o significado do que queriam exprimir. Por exemplo, constatou-se que um aluno classificado na categoria visual, ao fazer uma descrição da sua resolução referiu "x toma o valor de y", complementando esta descrição com a fórmula "x=x+y". Pela sua descrição textual a sua resolução estaria errada, no entanto a fórmula por ele estabelecida correspondia exactamente ao pedido.

Os alunos desta amostra encontravam-se quase que igualmente repartidos pela dimensão Activa/Reflexiva. Porém a maioria dos activos era-o em intensidade forte ou moderada, enquanto que a maioria dos reflexivos era-o em intensidade fraca ou moderada. Notaram-se também diferenças entre os indivíduos "activos e visuais" e os "reflexivos e visuais" constatando-se que:

- Os "activos e visuais", com uma forte componente activa, na sua maioria começavam por desenhar a resposta que depois complementavam com pouco texto ou respondiam unicamente com gráficos ou ilustrações.

- Os "reflexivos e visuais", com uma forte componente reflexiva, respondiam em primeiro lugar de forma textual e depois finalizavam a resposta com ilustrações.

A maioria dos alunos testados era sequencial, embora os valores apresentados fossem fracos ou moderados, nomeadamente 1, 3 e 5 em proporções sensivelmente iguais. Geralmente, estes alunos descreviam as suas respostas de forma mais detalhada, mostrando os diferentes passos, enquanto que os globais "saltavam" algumas etapas nas suas resoluções. Os sequenciais apresentaram dificuldade em generalizar soluções para os problemas. Constatou-se, em diversos problemas, que os sequenciais traduziam as suas soluções em mais etapas, muitas vezes menos eficientes, comparativamente com os globais.

Na generalidade das questões de todos os testes, constatou-se que os globais geralmente apresentavam um de dois tipos de comportamentos. Tinham classificações muito elevadas nas suas respostas ou então não respondiam às questões, o que para nós constituiu um indício de que precisavam de um tempo mais alargado para conseguir entender/responder às questões, mas quando o conseguiam, faziam-no na sua plenitude. Outro aspecto que merece ser salientado foi o facto de frequentemente os sequenciais conseguirem identificar comportamentos de processos isolados, nas diferentes iterações de um problema, mas revelando-se incapazes de determinar o seu resultado final ou global.

A maioria dos alunos testados era moderada ou fortemente sensorial. Os intuitivos eram-no em fraca medida. Apenas um aluno foi considerado moderamente intuitivo (7), que curiosamente foi dos que obteve melhores resultados na generalidade das questões, de todos os testes, destacando-se particularmente em questões que exigiam bastante agilidade mental e descoberta de enigmas, raciocínios errados, entre outros. Também de acordo com Kuri (2004) os intuitivos destacam-se no domínio de novos conceitos e sentem-se mais confortáveis que os sensoriais no que concerne às abstracções e formulações matemáticas. Também nós corroboramos esta opinião, pois a maioria dos alunos da amostra categorizados como sensoriais resolvia muitas vezes os problemas com métodos estabelecidos, baseando-se mais na memorização e repetição, apresentando fraca capacidade de abstracção.

Embora se tenha verificado que a maior parte dos alunos da amostra integrava as categorias visual, sensorial, activo e sequencial, este estudo não nos permite afirmar que existe um perfil de estilos de aprendizagem entre os alunos com dificuldades a programação. Isto porque, como confirmado noutros estudos, também a maioria dos alunos bem sucedidos a programação apresenta este perfil, tal como os restantes alunos de engenharia.

Refira-se ainda que o propósito das sessões era não apenas a de identificar défices de resolução de problemas, mas também o de proporcionar algum treino neste domínio, verificando a sua influência na aprendizagem de programação. Porém, os resultados obtidos no teste final de programação foram novamente bastante desanimadores, parecendo que o treino de resolução de problemas não gerou qualquer tipo de influência. Em termos estatísticos esses resultados eram impossíveis de provar, dado o tamanho insuficiente da amostra. Porém, realizou-se uma análise intuitiva, caso a caso, e não se notou qualquer tipo de melhoria nos exercícios de programação apresentados. Por um lado, pensamos que o treino não terá sido suficiente para ter reflexos positivos significativos em qualquer dos domínios (capacidade de resolução de problemas ou de programação). Por outro lado, também pensamos que estes resultados significam que as capacidades de resolução de problemas, embora possam constituir uma condição necessária para que um indivíduo seja bem sucedido na aprendizagem de programação, não são necessariamente suficientes para que tal aconteça.

Um resultado que se pode afirmar como realmente positivo esteve relacionado com os problemas de lógica ou que tinham alguma cilada. Estes eram os exercícios preferidos dos alunos. Notou-se que ao longo das sessões os alunos passaram a estar mais concentrados para detectar enigmas e passaram a conseguir resolvê-los na sua plenitude. Notou-se ainda que os alunos se entusiasmavam por esse tipo de exercícios, passando a procurá-los, especialmente na *world wide web*, trazendo muitas vezes desafios para os professores resolverem.

6.2.5 Conclusões

Este estudo permitiu-nos recolher as conclusões apresentadas na tabela 6.1.

Estudo A - Conclusões	
Resolução de Problemas	Os alunos não dominavam conceitos matemáticos básicos.
	Os alunos apresentaram dificuldades de cálculo.
	Os alunos tinham dificuldade em traduzir a descrição textual de um problema para uma fórmula matemática que o resolvesse.
	Os alunos desconheciam aspectos relevantes sobre figuras geométricas.
	Os alunos tinham dificuldades de interpretação do enunciado.
	Os alunos eram incapazes de definir critérios de comparação.
	Os alunos eram incapazes de se orientar no plano cartesiano.
	Os alunos desconheciam conceitos trigonométricos.
	Os alunos revelaram-se incapazes de aplicar conceitos trigonométricos.
	Os alunos tinham fracos níveis de abstracção.
	Os alunos tinham falta de treino lógico e de raciocínio.
Estilos de aprendizagem	O perfil de estilos de aprendizagem mais frequente era **Sensorial-Visual-Activo-Reflexivo**.
	Os visuais obtiveram melhores desempenhos nos exercícios apresentados através de figuras.
	Os visuais expressaram-se melhor através de ilustrações do que através de descrições textuais.
	Os visuais utilizaram mais representações esquemáticas nas suas resoluções do que os verbais.
	Os sequenciais detalharam mais as suas respostas, por vezes com alguma repetição ou ineficiência, enquanto que os globais omitiam os detalhes, por vezes necessários.
	Os sensoriais apresentaram fraca capacidade de abstracção.

Tabela 6.1: Conclusões do Estudo A

Estas conclusões indiciam a veracidade dos factores F1.1 – Muitos alunos com dificuldades de aprendizagem de programação apresentam muitas dificuldades em resolver problemas e F1.2 – Muitos alunos com dificuldades de aprendizagem de programação apresentam défices de conhecimentos matemáticos e lógicos. Apesar de com este estudo não podermos confirmar o factor F2.1 – A Engenharia Informática/Ciências da Computação atrai alunos com determinado perfil de estilos de aprendizagem, constatámos a existência de um

perfil de estilos de aprendizagem entre os alunos envolvidos (todos com dificuldades a programação).

6.3 Estudo B

Este estudo teve várias finalidades (Gomes e Mendes, 2008). Um dos objectivos foi o de validar a caracterização dos estilos de aprendizagem dos alunos de cursos superiores de Engenharia Informática (F2.1) e simultaneamente analisar a correlação entre os estilos de aprendizagem e os seus resultados à disciplina introdutória de programação (F2.2). Pretendeu-se também estudar os factores (F1.1 e F1.2) relativos às competências dos alunos em termos de resolução de problemas e competências matemáticas.

6.3.1 Contexto

Este estudo decorreu durante o 1º semestre do ano lectivo de 2007/2008 e envolveu dois grupos diferentes de estudantes. Um dos grupos incluiu alunos matriculados na Licenciatura em Engenharia Informática e de Sistemas, do Instituto Superior de Engenharia de Coimbra (LEIS-ISEC). O outro grupo incluiu alunos matriculados na Licenciatura em Engenharia Informática, da Universidade de Coimbra (LEI-UC). Todos os estudantes envolvidos se encontravam pela primeira vez no 1º ano. O grupo de alunos da LEIS-ISEC na sua 1ª disciplina de programação estudou a linguagem C, enquanto que o grupo de alunos da LEI-UC estudou Python. A amostra incluiu 87 alunos no grupo LEIS-ISEC, 6 do sexo feminino e os restantes do sexo masculino, com idades compreendidas entre os 17 e 40 anos. Neste grupo, a média das idades era de 21,08 anos, mas a mediana situava-se nos 19 anos, a idade comum para alunos do 1º ano. No outro grupo, LEI-UC, a amostra era de 102 alunos, 21 do sexo feminino e 81 do sexo masculino, com idades compreendidas entre os 17 e 41 anos, a média das idades era de 18,75 anos e a mediana situava-se nos 18 anos. A baixa percentagem de indivíduos do sexo feminino, em ambos os grupos, fez com que optássemos por não fazer distinção por género, nesta experiência.

6.3.2 Instrumentos e metodologia

Os instrumentos de pesquisa utilizados foram três: o teste *Index of Learning Styles* (Anexo G), um teste de diagnóstico (Anexo B.1) e um inquérito (Anexo B.2). O ILS, serviu para caracterizar os estilos de aprendizagem dos alunos. O inquérito tinha como objectivo recolher dados demográficos, alguns dados académicos e informação referente a conhecimentos prévios dos alunos. O teste de diagnóstico serviu para avaliar a capacidade

de resolução de problemas de base matemática e lógica. Estas capacidades foram testadas através de duas perguntas, sendo a escolha dos exercícios para integrar essas perguntas feita com base em défices observados na experiência anterior (Estudo A). A 1ª pergunta do teste de diagnóstico (Anexo B.1) envolvia cálculos matemáticos simples do tipo dos que são necessários para resolver problemas do quotidiano. Apresentámos este tipo de questão por pensarmos que muitos alunos são mal sucedidos com a aprendizagem de programação também devido à falta de competências de resolução problemas deste domínio. A 2º pergunta do teste de diagnóstico (Anexo B.1) consistiu num problema, adaptado de Epp (1990) e tentou testar a capacidade de resolução de problemas lógicos.

O teste de diagnóstico e o inquérito foram respondidos através de um procedimento tradicional, usando caneta e papel e o teste ILS foi respondido *on-line*. Os instrumentos de pesquisa foram aplicados em sala de aula, no início do ano lectivo, em horários previamente combinados entre a investigadora e os professores das disciplinas introdutórias de programação das duas instituições envolvidas, Algoritmos e Programação (AP) (da LEIS-ISEC) e Introdução à Programação e Resolução de Problemas (IPRP) (da LEI-UC).

Utilizámos análises quantitativas descritivas e de correlações entre os aspectos questionados considerados de interesse e os resultados às disciplinas introdutórias de programação.

6.3.3 Análise de resultados

Começamos a análise dos resultados deste estudo, com um aspecto que considerámos fundamental, a caracterização da amostra em termos da experiência prévia dos alunos a programação. Os resultados são apresentados na tabela 6.2 e na tabela 6.3.

	C	**Pascal**	**Java**	**Python**	**Outras**
Nenhum	51.16%	54.65%	87.21%	94.19%	76.47%
Básico	30.23%	25.58%	9.30%	5.81%	9.41%
Médio	16.28%	18.61%	3.49%	0%	12.94%
Avançado	2.33%	1.16%	0%	0%	1.18%

Tabela 6.2: Amostra LEIS-ISEC – Nível de conhecimentos em linguagens de programação

	C	**Pascal**	**Java**	**Python**	**Outras**
Nenhum	75.24%	76.19%	89.52%	89.52%	77.14%
Básico	12.38%	14.38%	10.48%	10.48%	9.52%
Médio	9.52%	4.67%	0%	0%	10.48%
Avançado	2.86%	4.76%	0%	0%	2.86%

Tabela 6.3: Amostra LEI-UC – Nível de conhecimentos em linguagens de programação

De notar que não foi realizado qualquer tipo de teste para aferir estes dados, mas antes foram resultado de uma das perguntas do inquérito onde se pedia ao aluno para autoclassificar o seu nível de conhecimento em termos de linguagens de programação (nenhum, básico, médio ou avançado).

Apesar da experiência prévia em determinado assunto nos parecer ter uma influência óbvia nos resultados obtidos a qualquer disciplina que inclua esse assunto, considerámos este aspecto particularmente relevante a programação. Porém, comprovámos este factor aplicando o teste do Qui-quadrado entre as variáveis experiência prévia a programação e as classificações finais a programação. Este teste permitiu obter probabilidades de significância $p<=0,05$, em ambas as amostras, rejeitando a hipótese de que as variáveis eram independentes. Assim, comprovou-se estatisticamente que havia associação entre as variáveis experiência prévia a programação e as classificações finais a programação.

Face ao exposto considerámos dois grupos, os alunos que considerámos não saber programar (os que indicaram nenhuns conhecimentos ou conhecimentos básicos de programação) e os alunos que considerámos ter alguma experiência prévia de programação (os que indicaram ter conhecimentos médios ou avançados de programação). A tabela 6.4 apresenta a experiência prévia dos alunos a programação, nas duas amostras estudadas.

	LEIS-ISEC	**LEI-UC**
Sem experiência	67,82%	85,29%
Com experiência	32,18%	14,71%

Tabela 6.4: Amostras LEIS-ISEC e LEI-UC – Nível de experiência a programação

Face ao descrito decidimos concentrar o nosso estudo nos alunos que indicaram não ter experiência prévia de programação ou ter uma experiência considerada básica. Assim, dos 87 alunos do grupo LEIS-ISEC, considerámos apenas 59 estudantes para a nova amostra e dos 102 alunos do grupo LEI-UC, considerámos apenas 87 alunos para a nova amostra. A diminuta quantidade de elementos do sexo feminino, em qualquer das amostras, levou a que não se fizesse qualquer distinção entre sexos.

6.3.3.1 Análise do factor F2.1

Este aspecto foi testado através de uma análise quantitativa descritiva. Constatámos que no grupo LEIS-ISEC a maioria dos alunos era sensorial (79,27%), visual (96,55%), activo (67,07%) e sequencial (78,83%). No grupo LEI-UC foi encontrada uma distribuição similar, a maioria dos alunos era sensorial (73,33%), visual (82,35%), activo (67,78%) e sequencial (65,91%). A mesma tendência é verificada noutros estudos encontrados na literatura (Rosati,

1996; Constant, 1997; Paterson, 1999; Rosati, 1999; Buxeda et al., 2001; De Vita, 2001; Livesay et al., 2002; Lopes, 2002; Kuri e Truzzi, 2002; Seery et al., 2003; Zywno, 2003).

A figura 6.2 ilustra a quantificação de cada categoria das diversas dimensões dos estilos de aprendizagem, na amostra LEIS-ISEC.

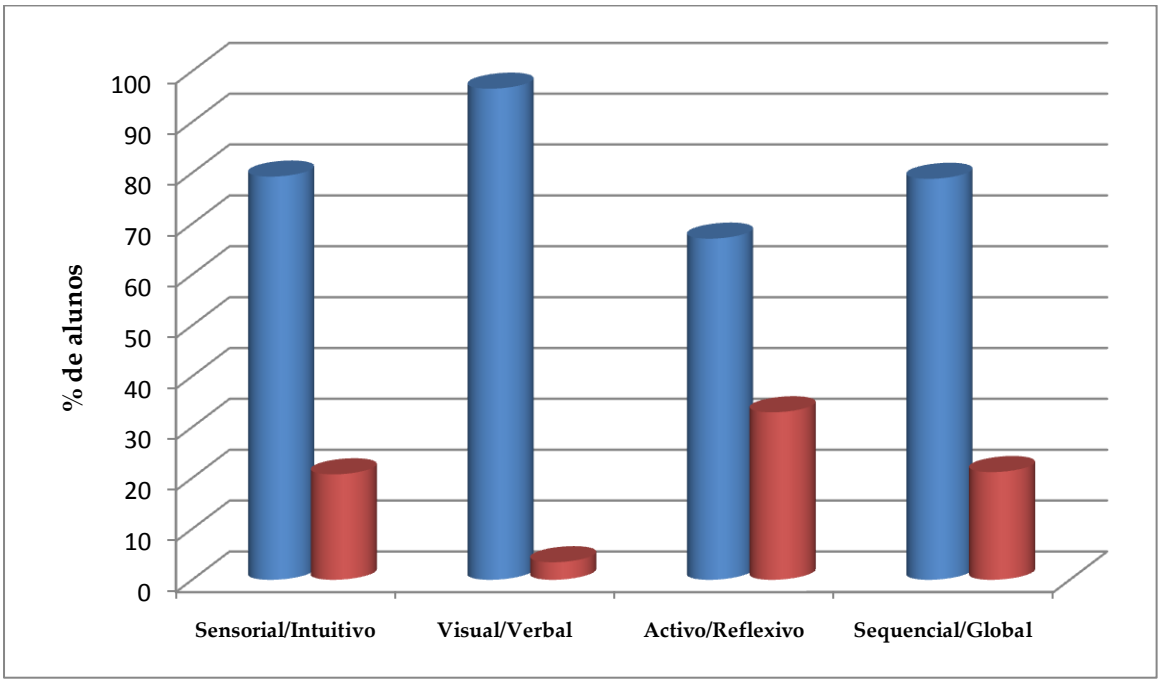


Figura 6.2: Amostra LEIS-ISEC – Estilos de aprendizagem

A tabela 6.5 ilustra a intensidade de cada categoria dos estilos de aprendizagem da amostra LEIS-ISEC.

Sensorial (79,27%)			**Intuitivo (20,73%)**		
Fraco (1-3)	Moderado (5-7)	Forte (9-11)	Fraco (1-3)	Moderado (5-7)	Forte (9-11)
53,85%	41,54%	4,61%	70,59%	23,52%	5,89%
Visual (96,55%)			**Verbal (3,45%)**		
Fraco (1-3)	Moderado (5-7)	Forte (9-11)	Fraco (1-3)	Moderado (5-7)	Forte (9-11)
37,98%	31,38%	30,64%	66,67%	0%	33,33%
Activo (67,07%)			**Reflexivo (32,93%)**		
Fraco (1-3)	Moderado (5-7)	Forte (9-11)	Fraco (1-3)	Moderado (5-7)	Forte (9-11)
52,63%	35,09%	12,28%	78,57%	21,43%	0%
Sequencial (78,83%)			**Global (21,17%)**		
Fraco (1-3)	Moderado (5-7)	Forte (9-11)	Fraco (1-3)	Moderado (5-7)	Forte (9-11)
51,97%	40,27%	7,76%	57,89%	42,11%	0%

Tabela 6.5: Amostra LEIS-ISEC – Estilos de aprendizagem

A figura 6.3 ilustra a quantificação de cada categoria das diversas dimensões dos estilos de aprendizagem, na amostra LEI-UC.

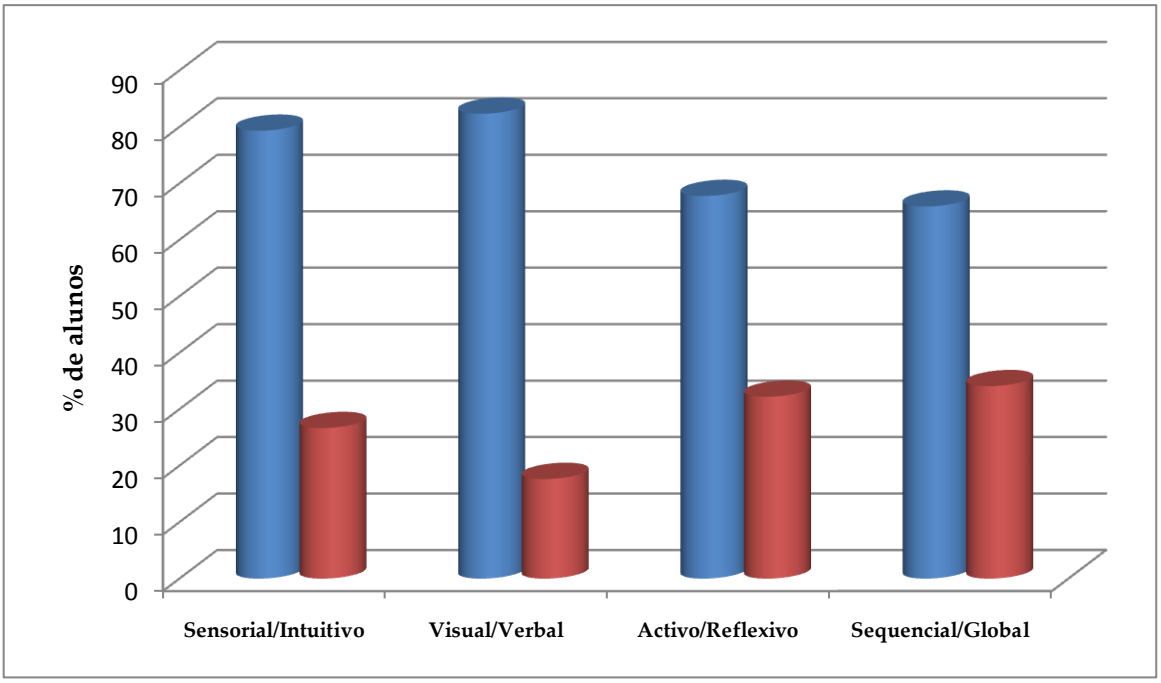


Figura 6.3: Amostra LEI-UC – Estilos de aprendizagem

A tabela 6.6 ilustra a intensidade de cada categoria dos estilos de aprendizagem da amostra LEI-UC.

Sensorial (73,33%)			**Intuitivo (26,67%)**		
Fraco (1-3)	Moderado (5-7)	Forte (9-11)	Fraco (1-3)	Moderado (5-7)	Forte (9-11)
56,05%	34,85%	9,10%	79,17%	20,83%	0%
Visual (82,35%)			**Verbal (17,65%)**		
Fraco (1-3)	Moderado (5-7)	Forte (9-11)	Fraco (1-3)	Moderado (5-7)	Forte (9-11)
26,58%	43,04%	30,38%	70%	30%	0%
Activo (67,78%)			**Reflexivo (32,22%)**		
Fraco (1-3)	Moderado (5-7)	Forte (9-11)	Fraco (1-3)	Moderado (5-7)	Forte (9-11)
47,54%	42,62%	9,84%	62,07%	37,93%	0%
Sequencial (65,91%)			**Global (34,09%)**		
Fraco (1-3)	Moderado (5-7)	Forte (9-11)	Fraco (1-3)	Moderado (5-7)	Forte (9-11)
74,14%	20,69%	5,17%	66,67%	33,33%	0%

Tabela 6.6: Amostra LEI-UC – Estilos de aprendizagem

Face ao exposto, podemos afirmar ser verdadeiro o factor F2.1 – A Engenharia Informática/Ciências da Computação atrai alunos com determinado perfil de estilos de aprendizagem. As tabelas e gráficos anteriores revelam que o perfil de estilos de aprendizagem predominante nas duas amostras de licenciaturas em Engenharia Informática é sensorial-visual-activo-sequencial, o que coincide com os resultados de diversos estudos na mesma área referidos no capítulo 3.

6.3.3.2 Análise do factor F2.2

Este factor foi estudado, através de uma análise quantitativa de correlações. Desta forma, tentámos obter correlações entre o desempenho obtido à primeira disciplina de programação e os diferentes perfis de estilos de aprendizagem. Porém, a identificação de todos os possíveis perfis, não deixaria um número possível de elementos para obter dados estatísticos, dada a possibilidade de coexistência de 16 perfis, sem considerar a intensidade de cada dimensão. Então, as análises de correlações foram realizadas entre cada uma das categorias de cada dimensão separadamente, nomeadamente sensorial/intuitivo, visual/verbal, activo/reflexivo e sequencial/global.

No grupo LEI-UC não foi encontrada qualquer correlação entre o desempenho a programação e as diferentes categorias dos estilos de aprendizagem. Também estudos conduzidos por Byrne e Lyons (2001), usando o modelo de Felder-Silverman e o seu ILS, revelaram a mesma ausência de correlação.

No grupo LEIS-ISEC apenas foi encontrada uma correlação negativa (r=-4,07 ao nível 0,05, *2–tailed*) verificada nos alunos que possuíam a categoria activa, significando que uma maior intensidade nesta categoria se traduziu num pior desempenho a programação. Este facto deveria também significar que quanto mais reflexivo fosse um aluno melhor seria o seu desempenho a programação. O facto de este aspecto não se verificar poderá estar relacionado com a percentagem de alunos reflexivos, insuficiente para obter correlações estatisticamente significativas. Outros estudos usando o ILS (Allert, 2004; Thomas et al., 2002) revelaram que os aprendizes reflexivos e verbais obtiveram melhores resultados a programação do que os activos e visuais.

De referir que, apesar de ambas as amostras conterem mais de 30 alunos, foram realizados os testes de normalidade, nomeadamente os testes de Kolmogorov-Smirnov e Shapiro-Wilk, atestando a normalidade das amostras, pressuposto para a aplicação dos testes de correlação.

Face ao exposto, não podemos afirmar ou rejeitar o factor F2.2, na medida em que numa amostra não se obteve correlação entre os estilos de aprendizagem dos alunos e os seus resultados à primeira disciplina de programação e noutra amostra obteve-se apenas uma correlação negativa relativamente a uma categoria dos estilos de aprendizagem.

6.3.3.3 Análise dos factores F1.1 e F1.2

A análise destes factores foi feita, através de análises quantitativas descritivas e de correlações, apresentadas de seguida. Para estas análises recorremos aos dados académicos e resultados dos testes de diagnóstico.

Primeiramente interessou-nos verificar a relação da nota de candidatura (Média de acesso ao curso) com os resultados à primeira disciplina de programação. A informação relativa à Média de ingresso no curso é apresentada na tabela 6.7 e na figura 6.4 e na figura 6.5. A escala utilizada é de 0 a 20.

	LEIS-ISEC	**LEI-UC**
N	51	86
Média	13,17	13,97
Mediana	12,7	13,73
Desvio Padrão	1,52	1,74
Mínimo	11,00	11,00
Máximo	17,00	19,20

Tabela 6.7: Amostras LEIS-ISEC e LEI-UC – Média de acesso ao curso

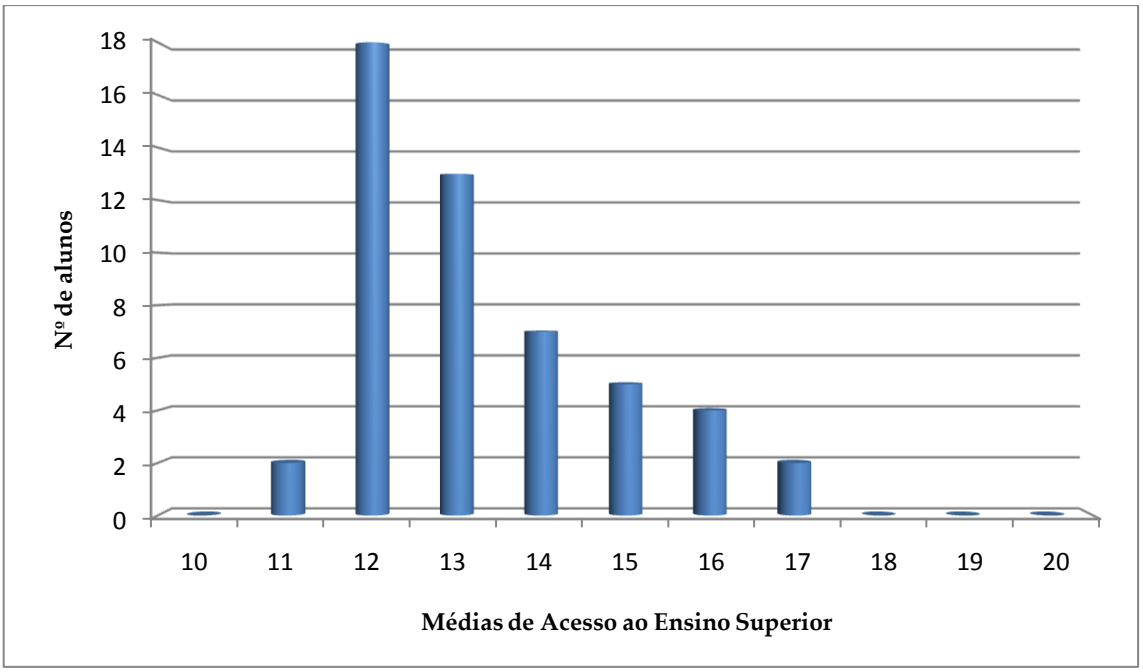


Figura 6.4: Amostra LEIS-ISEC – Distribuição das Médias de Acesso

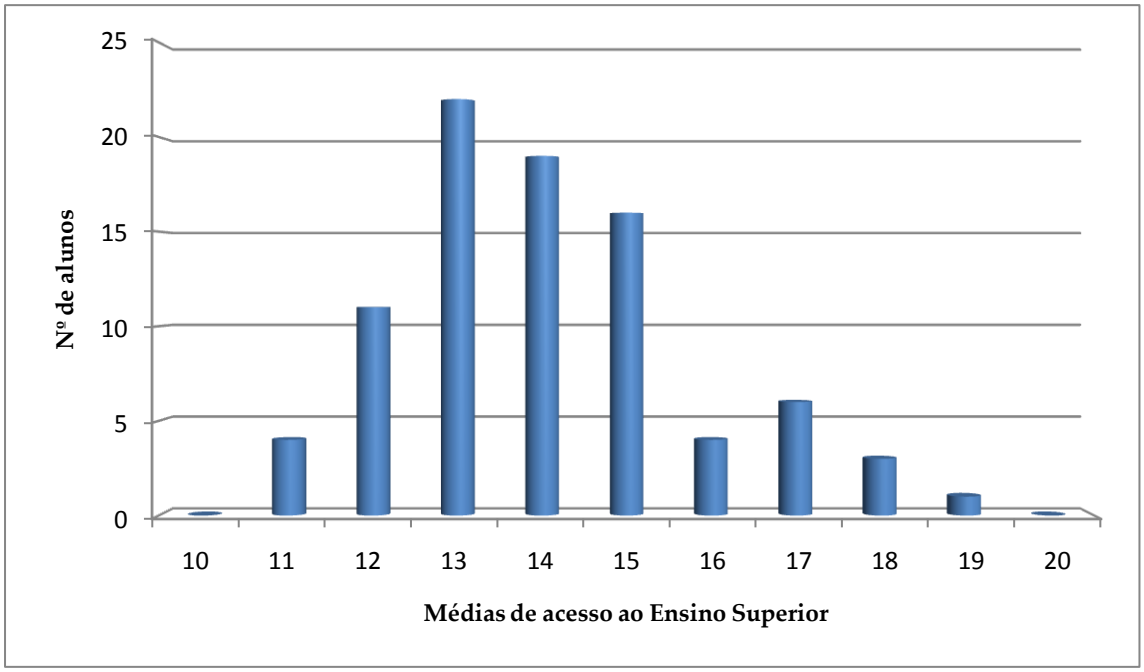


Figura 6.5: Amostra LEI-UC – Distribuição das Médias de Acesso

Pretendíamos saber se existia alguma correlação entre a média de acesso ao curso e a classificação obtida à primeira disciplina de programação. Foi encontrada uma correlação de r=0,282 ao nível de 0,05 (*2-tailed*) no grupo LEIS-ISEC e uma correlação mais forte de r=0,451 ao nível de 0,01 (*2-tailed*) no grupo LEI-UC. Estes resultados significam que os alunos com melhores médias de acesso ao ensino superior foram também os que obtiveram melhores resultados na primeira disciplina de programação, em qualquer dos grupos estudados. De referir que, foram realizados os testes de normalidade, nomeadamente os testes de Kolmogorov-Smirnov e Shapiro-Wilk, atestando a normalidade das amostras, pressuposto para a aplicação dos testes de correlação.

Outro aspecto que também achámos relevante foi o conhecimento das classificações obtidas pelos alunos na prova de acesso, numa das seguintes disciplinas de interesse: Matemática, Física ou Geometria Descritiva. Considerámos ser importante este tipo de conhecimento, por pensarmos que o desempenho dos alunos nestas disciplinas poderá ser usado como um indicador da capacidade de resolução de problemas em programação. Deste modo, cruzámos essas classificações com as classificações obtidas na primeira disciplina de programação e encontrámos algumas diferenças entre os grupos, veja-se a tabela 6.8.

	LEIS-ISEC	LEI-UC
N	48	85
Média	12,56	13,66
Mediana	13,00	14,00
Desvio Padrão	1,98	2,17
Mínimo	10,00	10,00
Máximo	18,00	18,00

Tabela 6.8: Amostras LEIS-ISEC e LEI-UC – Classificações na prova de acesso ao Ensino Superior numa disciplina de interesse

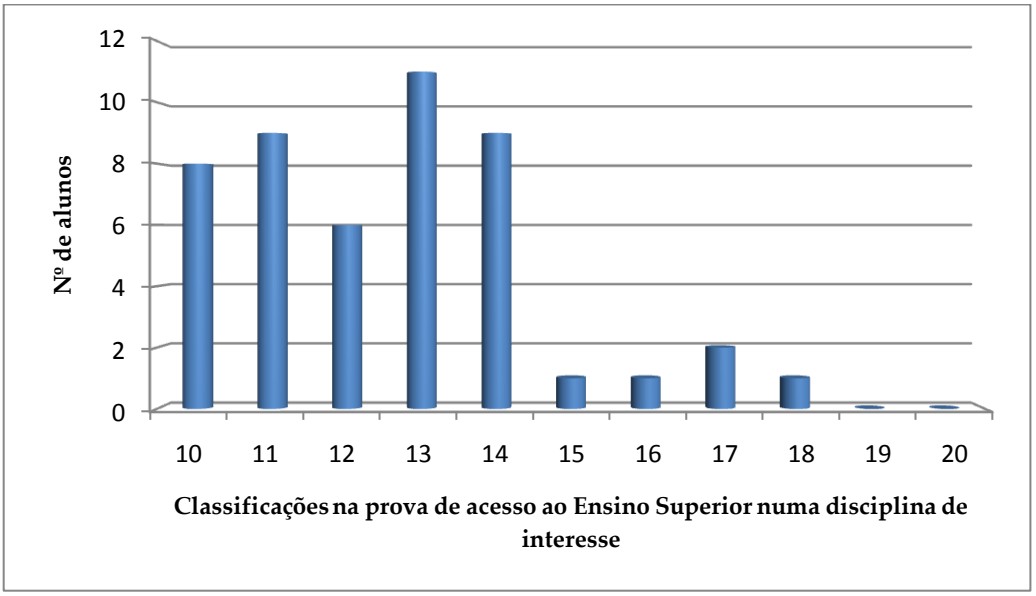


Figura 6.6: Amostra LEIS-ISEC – Distribuição das classificações na prova de acesso de uma disciplina de interesse

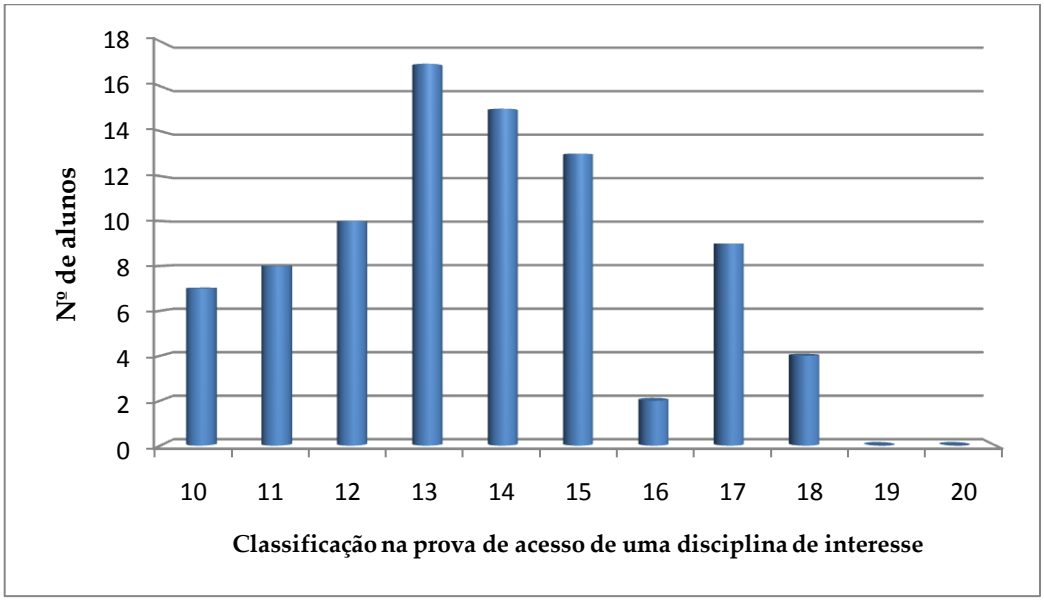


Figura 6.7: Amostra LEI-UC – Distribuição das classificações na prova de acesso de uma disciplina de interesse

No grupo LEIS-ISEC, não conseguimos encontrar qualquer correlação significativa. Fazendo uma análise caso a caso, verificámos inclusivamente algumas situações pontuais com indicações de correlações contrárias. Por exemplo, um aluno que obteve 18 valores a AP, obteve 11 valores na prova de Matemática. Por outro lado, houve alunos que obtiveram boas classificações a Matemática e reprovaram a AP. Por exemplo, 2 alunos com 15 valores em Matemática obtiveram 3 e 0 valores a AP. Quatro alunos com 17 valores a Matemática obtiveram 4, 5, 6 e 8 valores a AP. Um aluno com 16 valores a Matemática obteve 8 valores a AP. Estes últimos resultados são, na nossa perspectiva, muito prováveis de acontecer, pois pensamos que a capacidade matemática é apenas um dos factores que poderão condicionar o sucesso a uma disciplina de programação, mas existirão certamente outros. Adicionalmente, os conteúdos da prova de acesso a matemática, referentes aos últimos anos do ensino secundário poderão não ser os mais relevantes para um bom desempenho a programação. Neste grupo, tivemos alguns alunos que realizaram a prova de Geometria Descritiva em vez da prova de Matemática. Embora tenhamos tido poucos alunos nesta situação, de forma a obter validações estatísticas, existiu uma certa indicação de que poderia haver alguma correlação positiva entre a capacidade de aprender a programar e o desempenho a Geometria Descritiva. Por exemplo, 2 alunos com 17 e 19 valores a Geometria Descritiva obtiveram 16 valores a AP. Talvez este resultado possa ser intuitivamente interpretado como a capacidade de abstracção envolvida nos dois tipos de matérias (Programação e Geometria Descritiva) ser semelhante.

Porém, no grupo LEI-UC foi obtida uma correlação de r=0,373 ao nível de 0,01 (*2-tailed*) entre as classificações obtidas na prova de Matemática (todos os alunos indicaram esta disciplina) e os resultados a IPRP. Pelo que pudemos analisar, pensamos que a diferença possa ser explicada, essencialmente pelo tipo de exames e actividades desenvolvidas em IPRP que terão uma índole matemática mais forte.

Relativamente ao teste de diagnóstico, as classificações obtidas nas duas questões foram muito diferentes, porém, similares nos dois grupos, como pode ser observado na tabela 6.9, na tabela 6.10, na figura 6.8 e na figura 6.9.

	LEIS-ISEC	**LEI-UC**
N	46	69
Média	26,30%	19,06%
Mediana	15,00%	5,00%
Desvio Padrão	28,72%	29,72%
Mínimo	0%	0%
Máximo	100%	100%

Tabela 6.9: Amostras LEIS-ISEC e LEI-UC – Classificação da pergunta 1 do Teste de Diagnóstico

	LEIS-ISEC	LEI-UC
N	56	78
Média	70,80%	75,90%
Mediana	95,00%	90,00%
Desvio Padrão	37,67%	30,34%
Mínimo	0%	0%
Máximo	100%	100%

Tabela 6.10: Amostras LEIS-ISEC e LEI-UC – Classificação da pergunta 2 do Teste de Diagnóstico

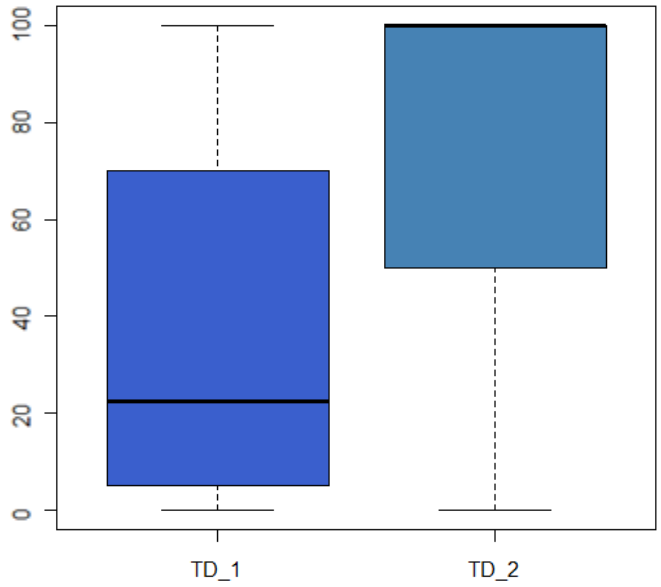


Figura 6.8: Amostra LEIS-ISEC – Classificação das perguntas 1 e 2 do Teste de Diagnóstico

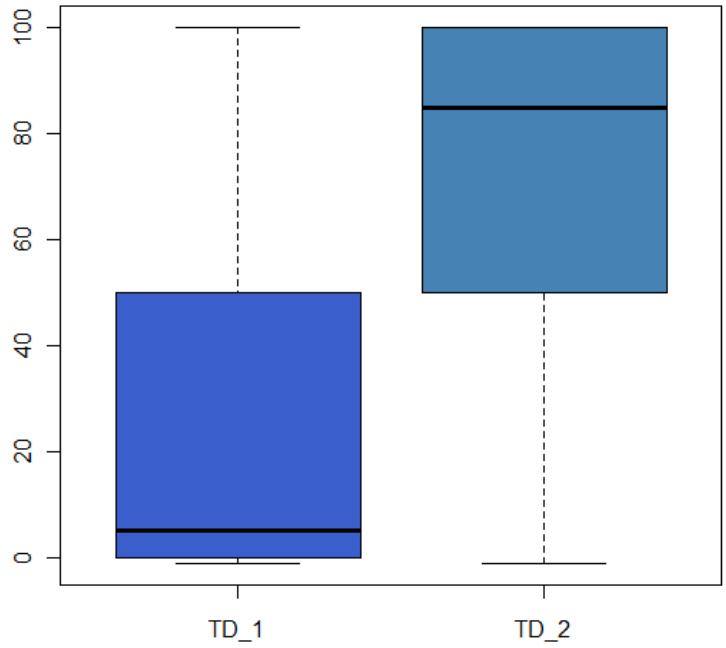


Figura 6.9: Amostra LEI-UC – Classificação das perguntas 1 e 2 do Teste de Diagnóstico

Como evidenciado, muitos alunos obtiveram notas muito baixas na 1ª questão e a maioria deles obteve positiva ou até mesmo boas classificações na 2ª questão, em ambas as amostras (LEIS-ISEC e LEI-UC). De notar que a 1ª questão envolvia a capacidade de cálculo associada à obtenção de uma fórmula para resolver um problema. A 2ª questão era essencialmente do foro lógico.

Posteriormente cruzámos estas classificações com as classificações obtidas a AP e IPRP. Encontrámos uma forte correlação entre as classificações obtidas na 1ª questão do teste de diagnóstico e as classificações obtidas a programação, em ambos os grupos, nomeadamente, r=0,492 ao nível de 0,01 (*2-tailed*) no grupo LEIS-ISEC e r=0,416 ao nível de 0,05 (*2-tailed*) no grupo LEI-UC. Esta correlação significa que os alunos que obtiveram melhores resultados a AP e IPRP foram também aqueles com melhores classificações na pergunta que envolvia cálculos matemáticos. Porém, para a 2ª questão não foi encontrada qualquer correlação. Estes resultados sugerem-nos que a programação está relacionada com a capacidade de abstracção envolvida em cálculos similares àqueles requeridos para responder à 1ª questão do teste de diagnóstico. De notar que as respostas a esta questão foram classificadas tendo em conta a capacidade de abstracção. Ou seja, pretendia-se que as respostas dos alunos fossem traduzidas através de uma fórmula/algoritmo genérico que pudesse funcionar para qualquer caso, atribuindo-se uma classificação baixa a respostas que se reportassem a exemplos específicos ou concretos. Também autores como Byrne e Lyons (2001) e Chmura (1998) encontraram uma forte correlação positiva entre o desempenho dos alunos a uma determinada disciplina de programação e o seu desempenho noutras disciplinas que envolviam a resolução de problemas. Apesar de os alunos precisarem de ser capazes de pensar logicamente e de ter capacidades de cálculo, os resultados indiciam que os alunos têm o raciocínio lógico mais desenvolvido do que as capacidades de cálculo ou que o tipo de exercícios usados nestas disciplinas de programação estão mais relacionados com as capacidades de cálculo do que com as questões puramente lógicas.

Também correlacionámos as classificações obtidas pelos alunos na prova de acesso numa das disciplinas de Matemática, Física ou Geometria Descritiva, com as questões do teste de diagnóstico. Porém, não encontrámos correlações entre elas. No entanto, fizemos uma análise intuitiva, concentrando-nos especialmente nos resultados obtidos na prova de Matemática (a realizada pela maioria dos alunos) e nos resultados da 1ª pergunta do teste de diagnóstico. Encontraram-se situações de alunos com baixas classificações nesta questão do teste de diagnóstico que obtiveram elevadas classificações na prova de Matemática, bem como situações contrárias. Consideramos que estes factos sugerem que as questões que envolvem cálculos possam estar mais relacionadas com as competências necessárias para programar do que o tipo de competências requeridas pela disciplina de Matemática nos

últimos anos do ensino secundário. Este facto vem também reforçar a nossa ideia sobre o tipo de competências matemáticas necessárias para as disciplinas iniciais de programação.

Pensamos poder considerar verdadeiro o factor F1.1 – Muitos alunos com dificuldades de aprendizagem de programação apresentam muitas dificuldades em resolver problemas, face á correlação obtida entre a 1ª pergunta do teste de diagnóstico e os desempenhos obtidos a programação, traduzidos pelas classificações finais na disciplina introdutória de programação, em ambas as amostras.

Porém, esta experiência não permitiu comprovar completamente o factor F1.2 – Muitos alunos com dificuldades de aprendizagem de programação apresentam défices de conhecimentos matemáticos e lógicos. Relativamente à correlação entre as bases matemáticas, traduzidas através da classificação obtida na prova de acesso a Matemática, e os desempenhos a programação, os estudos confirmam este aspecto numa amostra, mas noutra não comprovam qualquer correlação. Como referido cremos que a diferença possa ser explicada pelo tipo de exames e actividades com um cariz matemático mais ou menos forte, em cada uma das disciplinas introdutórias de programação. Relativamente aos aspectos de raciocínio lógico, avaliados na 2ª questão do teste de diagnóstico não foi obtida qualquer correlação com os resultados às disciplinas introdutórias de programação, em qualquer das amostras.

6.3.3.4 Análise dos aspectos motivacionais

Apesar de não ser considerado inicialmente factor de investigação, a realização de várias experiências e o contacto de alguma proximidade com os alunos começou a suscitar outro tipo de conjecturas relacionadas com a motivação dos alunos para a área de estudo. Desta forma, o inquérito tentou também avaliar este tipo de factores. Para o estudo dos aspectos motivacionais usámos duas perguntas relacionadas com o processo utilizado para a candidatura do aluno ao ensino superior. No formulário de candidatura os alunos podem indicar 6 pares de curso/instituição numa ordem que vai do mais ao menos desejado. Desta forma, perguntámos ao aluno qual a ordem em que colocou o par curso/instituição em que se encontra. A maioria (60,98% no grupo LEIS-ISEC e 89,98% no grupo LEI-UC) respondeu que foi a sua primeira opção, como ilustrado na tabela 6.11.

	LEIS-ISEC	**LEI-UC**
1ª opção	60,98%	89,98%
Outra opção	39,02%	10,02%

Tabela 6.11: Amostras LEIS-ISEC e LEI-UC – Opção de escolha do curso

Porém, a razão pela qual os estudantes fazem escolhas está muitas vezes relacionada não somente com uma preferência pessoal, mas principalmente com as classificações conseguidas, e consequentemente com os cursos onde provavelmente serão aceites com base nessas classificações. Assim, tentámos isolar este factor usando uma segunda pergunta. Para tal, pedimos que o aluno imaginasse que tinha a classificação máxima (20 valores), e nessa situação indicasse o curso/instituição que escolheria em primeiro lugar. Houve uma diversidade enorme de respostas que classificámos em três grupos. Aquele correspondente ao grupo de alunos que escolheriam o mesmo par curso/instituição (os quais considerámos altamente motivados). O grupo correspondente aos alunos que escolheriam o mesmo curso mas numa instituição diferente (os quais considerámos moderadamente motivados) e aqueles que escolheriam um curso completamente diferente (os quais considerámos com baixa motivação para a área de estudo). Os resultados desta divisão, para ambos os grupos, encontram-se representados na figura 6.10 e na figura 6.11.

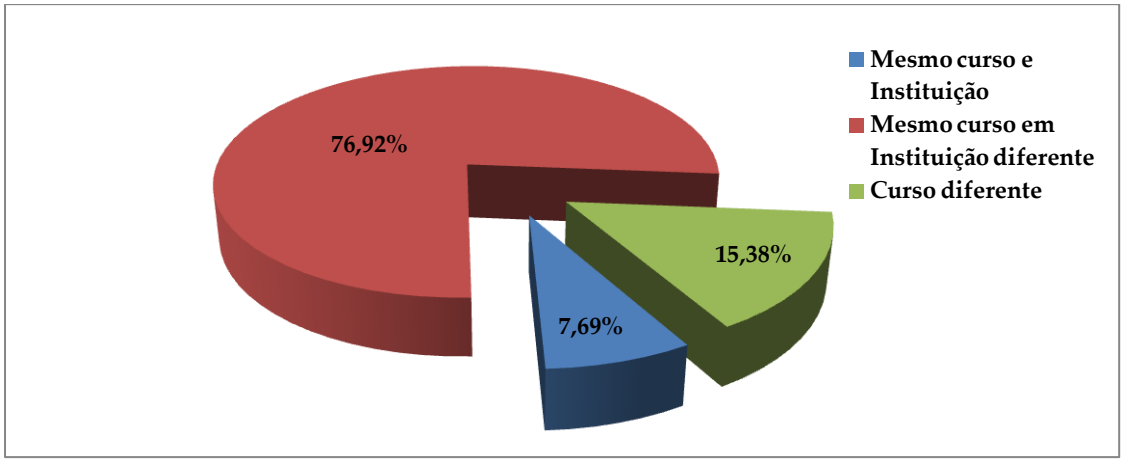


Figura 6.10: Amostra LEIS-ISEC – Escolha do curso ideal

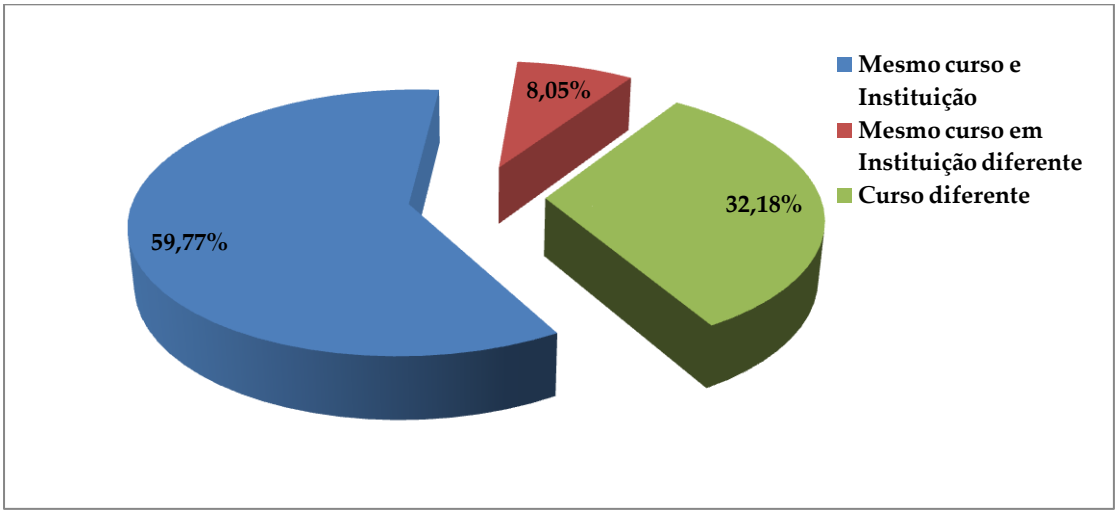


Figura 6.11: Amostra LEI-UC – Escolha do curso ideal

A análise da informação dos gráficos anteriores indicia que a maioria significativa dos alunos da LEIS-ISEC se encontram na área de estudo ideal, se bem que o desejassem fazer noutra instituição. Relativamente à LEI-UC, apesar da maioria dos alunos se encontrar a frequentar o curso pretendido na instituição desejada, existe um número considerável de alunos que gostaria de estar noutra área diferente.

Cruzámos ainda esta informação com as classificações obtidas nas disciplinas de programação em estudo, mas não encontrámos qualquer correlação, estatística ou intuitiva. Uma análise mais detalhada mostrou inclusivamente situações de alunos que não pareciam estar motivados, de acordo com a nossa classificação, mas que obtiveram boas notas a programação, bem como o contrário, em ambos os grupos (LEIS-ISEC e LEI-UC).

Usámos ainda uma terceira pergunta para tentar avaliar o tipo de motivação do aluno, com base nos tipos definidos em Jenkins (2001), a saber motivação extrínseca, motivação social e motivação de realização. Foi ainda usada uma outra categoria, correspondendo "à motivação nula", para acomodar os casos que pudessem estar fora daquelas categorias. Para avaliar este aspecto incluímos a seguinte pergunta:

Qual das seguintes afirmações descreve a sua atitude face ao curso em que ingressou?

a) Eu quero obter bons resultados para minha própria satisfação.

b) Eu quero obter bons resultados para satisfazer os meus pais, família e amigos.

c) Eu quero obter bons resultados para satisfazer os meus professores.

d) Eu quero obter bons resultados de modo a ter um bom emprego.

e) O meu objectivo principal é passar.

Apesar de nesta pergunta ser pedido aos alunos para assinalarem apenas uma resposta, correspondente à afirmação que mais reflectia a sua atitude perante o curso, a maioria dos alunos assinalou mais do que uma resposta. Se considerássemos apenas os alunos que escolheram uma única resposta não seria possível o tratamento estatístico desta questão, pelo que considerámos separadamente o número de respostas a cada uma das alíneas. A escolha a) reflectia a motivação de realização, as escolhas b) e c) traduziam motivações sociais, a selecção d) representava a motivação extrínseca e a e) a motivação nula. Os resultados são mostrados na tabela 6.12.

	LEIS-ISEC	LEI-UC
Motivação de realização	42,37%	40,23%
Motivação social	5,17%	2,30%
Motivação extrínseca	58,62%	45,98%
Motivação nula	0%	1,15%

Tabela 6.12: Amostras LEIS-ISEC e LEI-UC – Tipo de motivação

Os tipos de motivação podem condicionar os resultados de aprendizagem, estando normalmente as motivações intrínsecas, relacionadas com o desejo real de aprender, associadas a aprendizagens profundas e consequentemente a melhores desempenhos e resultados académicos. Os resultados deste estudo indiciam que os alunos se encontram bastante motivados, pois mais de metade dos alunos apresenta motivação intrínseca e a motivação nula é indicada num número de respostas nulas ou desprezáveis.

Correlacionámos os resultados obtidos nas disciplinas de programação em estudo com o tipo de motivação definido anteriormente, porém, não encontrámos qualquer tipo de correlação. De notar porém, que nalguns casos o número de respostas não era suficiente para obter resultados estatísticos significativos. Por outro lado, apesar de se ter indicado explicitamente a pertinência de os alunos indicarem apenas uma única resposta, muitos não o fizeram, tendo a maioria significativa escolhido várias opções. No entanto, fizemos uma análise pormenorizada, caso a caso e encontrámos alguns factos curiosos. Por exemplo, nenhum aluno escolheu a opção c). Entretanto, como esperado, todos os alunos com classificações mais elevadas a programação (pelo menos 16 valores) indicaram, como resposta à pergunta anterior a opção a) ou as opções a) e d). Este aspecto parece sugerir que os alunos que tiveram classificações mais elevadas a programação estivessem possivelmente mais motivados intrinsecamente. Porém, resultados semelhantes emergem dos alunos com maus resultados a programação

6.3.4 Conclusões

Este estudo permitiu-nos recolher as conclusões apresentadas na tabela 6.13 e na tabela 6.14.

	LEIS-ISEC	LEI-UC
Experiência de programação	r=1	r=1
Média Geral de ingresso	r=0,282*	r=0,451**
Nota de ingresso a Matemática	—	r=0,373**
Pergunta 1 (Teste de Diagnóstico)	r=0,492**	r=0,416*

*(*ao nível 0,05 (2-tailed), **ao nível 0,01 (2-tailed))*

Tabela 6.13: Amostras LEIS-ISEC e LEI-UC – Estatística das variáveis com influência nos desempenhos a programação

Estudo B - Conclusões	
Resolução de Problemas	Na amostra LEIS-ISEC foi encontrada correlação entre o desempenho a programação e a média de ingresso no ensino superior. Na amostra LEI-UC foi encontrada correlação entre o desempenho a programação e a média de ingresso no ensino superior. Na amostra LEIS-ISEC não foi encontrada correlação entre o desempenho a programação e as classificações à disciplina de Matemática. Na amostra LEI-UC foi encontrada correlação entre o desempenho a programação e as classificações obtidas à disciplina de Matemática. Na amostra LEIS-ISEC foi encontrada correlação entre o desempenho a programação e a classificação obtida na pergunta do teste de diagnóstico que envolvia cálculos matemáticos. No grupo LEI-UC foi encontrada correlação entre o desempenho a programação e a classificação obtida na pergunta do teste de diagnóstico que envolvia cálculos mateméticos. Não foi encontrada correlação entre o desempenho a programação e os níveis motivacionais dos alunos, em qualquer das amostras LEIS-ISEC ou LEI-UC.
Estilos de aprendizagem	O perfil de estilos de aprendizagem mais frequente foi **Sensorial-Visual-Activo-Reflexivo** na amostra LEIS-ISEC. O perfil de estilos de aprendizagem mais frequente foi **Sensorial-Visual-Activo-Reflexivo** na amostra LEI-UC.* Não foi encontrada correlação entre o desempenho a programação e as diferentes categorias dos estilos de aprendizagem, na amostra LEIS-UC. Foi obtida uma correlação negativa com o desempenho a programação dos alunos pertencentes à categoria activa, na amostra LEIS-ISEC.

Tabela 6.14: Conclusões do Estudo B

Em geral, pensamos poder afirmar que este estudo nos forneceu fortes indícios no sentido afirmar que:

- O factor F1.1 – "Muitos alunos com dificuldades de aprendizagem de programação apresentam muitas dificuldades em resolver problemas" é verdadeiro.
- O factor F1.2 – "Muitos alunos com dificuldades de aprendizagem de programação apresentam défices de conhecimentos matemáticos e lógicos" não foi provado na íntegra. Em certa medida pudemos confirmar os défices de

conhecimentos matemáticos, porém, as dificuldades de raciocínio lógico não foram verificadas, de forma explícita.

- O factor F2.1 – "A Engenharia Informática/Ciências da Computação atrai alunos com determinado perfil de estilos de aprendizagem" é verdadeiro, pois em ambos as amostras de estudo foram encontrados perfis de estilos de aprendizagem coincidentes e também conducentes com outros estudos encontrados na literatura, referentes ao mesmo grupo genérico de alunos.
- Não encontrámos indícios que nos permitam confirmar o factor F2.2 – "Existe correlação entre os estilos de aprendizagem dos alunos e os seus resultados à primeira disciplina de programação", na medida em que não foi obtida qualquer correlação significativa entre os perfis de estilos de aprendizagem e os resultados a programação.

6.4 Estudo C

Este estudo (Pacheco et al., 2008) teve como finalidade principal analisar o factor F2.3, relativamente ao desajustamento existente entre as actividades de programação normalmente propostas aos alunos e os seus níveis cognitivos. Para tal, pensou-se inicialmente em excluir a dificuldade associada à natureza matemática de muitos problemas de programação e estudar apenas as dificuldades do ponto de vista da programação. Assim, recorremos a alunos de um curso de Matemática em vez de alunos das Engenharias. As suposições iniciais eram que este grupo de alunos não teria dificuldades em resolução de problemas genéricos ou que incluíssem conceitos matemáticos implícitos ou explícitos, devido à natureza do curso em que ingressaram. Desta forma isolaríamos os problemas de natureza matemática e poder-nos-íamos concentrar exclusivamente nas dificuldades do ponto de vista de programação. Porém, para não assumirmos este dado como adquirido, estruturámos as experiências de forma a poder separar o tipo de dificuldades e em função disso, propor actividades mais adequadas ao nível cognitivo do aluno. Esta organização permitiu-nos estudar novamente o factor F1.2 – Muitos alunos com dificuldades de aprendizagem de programação apresentam défices de conhecimentos matemáticos e lógicos.

6.4.1 Contexto

Este estudo decorreu no Departamento de Matemática da Universidade de Coimbra, no 2º semestre de 2007/2008. O estudo incluiu a realização de duas sessões com um grupo de 7 alunos do curso de Matemática matriculados na disciplina introdutória de programação,

Métodos de Programação I (MP-I). Nesta disciplina é suposto que os alunos aprendam os conceitos básicos de programação, desenvolvendo pequenos programas usando a linguagem Pascal. Os alunos que integraram a experiência foram os voluntários obtidos de um grupo, aconselhado pela professora responsável por essa disciplina, por serem aqueles que apresentavam mais dificuldades, mostrando claros sinais de que o insucesso estava iminente.

6.4.2 Instrumentos

Os instrumentos de investigação foram constituídos por testes desenvolvidos pela investigadora. Para a sessão inicial foram preparados três testes, distribuídos aos alunos de acordo com as reacções de cada um ao teste anterior. O 1º teste incluía conceitos matemáticos e problemas de programação envolvendo esses conceitos. Os outros dois testes incidiam, um sobre conceitos matemáticos, outro sobre programação. Na 2ª sessão, foram distribuídos dois testes, um sobre aspectos puramente matemáticos e outro sobre programação envolvendo os conceitos matemáticos explorados no outro teste. Adicionalmente, nesta 2ª sessão, foi fornecida uma folha explicativa dos conceitos matemáticos em causa.

Os instrumentos de pesquisa foram aplicados em sala de aula, em horários previamente combinados entre a investigadora, os alunos participantes e a docente responsável pela disciplina de Métodos de Programação I. Inicialmente os alunos receberam uma breve explicação sobre a investigação e os seus objectivos após o que receberam os instrumentos e respectivas instruções.

6.4.3 Metodologia

A metodologia genérica utilizada neste estudo foi a observação, pelo facto de pretendermos um contacto mais directo com os alunos a fim de melhor identificar as suas dificuldades.

Na 1ª sessão, a investigadora ia fornecendo a cada aluno exercícios adequados ao seu nível de conhecimento, de acordo com as dificuldades demonstradas (de matemática ou de programação) no 1º teste (Anexo C.1 – Teste 1.1). Os exercícios deste teste basearam-se essencialmente no conceito de mínimo múltiplo comum, abordado primeiro conceptualmente e depois através de contextos de programação. A detecção do tipo de dificuldades neste 1º teste tornou possível guiar cada aluno para tarefas de acordo com os seus conhecimentos, permitindo usar uma abordagem taxonómica e personalizada. Desta forma, os alunos eram direccionados para dois outros testes um de conhecimentos matemáticos (Anexo C.2 – Teste 1.2) e outro de programação (Anexo C.3 – Teste 1.3), em função do tipo de dificuldades detectadas. A sequência das questões apresentadas nestes

dois testes seguiu sensivelmente a Taxonomia de Objectivos Educacionais de Bloom. A investigadora conduziu as experiências tentando interferir o mínimo possível, mas observando e intervindo, de uma forma sensata, para perceber as dúvidas dos alunos.

Na 2ª sessão foi distribuído primeiramente um teste de conceitos matemáticos (Anexo C.4 – Teste 2.1) e depois outro de programação (Anexo C.5 – Teste 2.2) envolvendo os mesmos conceitos matemáticos do 1º teste. Em qualquer dos testes as questões foram colocadas por ordem crescente de dificuldade cognitiva. Os testes estavam relacionados com conceitos matemáticos elementares.

Na 1ª sessão, o 1º teste (Teste 1.1) foi distribuído a todos os alunos e tinha como finalidade verificar se eles estavam familiarizados com o conceito de mínimo múltiplo comum (m.m.c.) e se eram capazes de o aplicar do ponto de vista conceptual e num contexto algorítmico. Na 1ª questão era pedido aos alunos que calculassem o m.m.c. entre dois números. A ideia era averiguar se os alunos conheciam e sabiam aplicar o conceito. Na questão 2 era pedido aos alunos para desenvolver um programa computacional que calculasse o m.m.c. entre dois números. O propósito desta questão era o de identificar se os alunos que conseguiam aplicar o conceito teórico numa situação concreta, também o conseguiam fazer para criar um algoritmo que o usasse. O objectivo era o de identificar os alunos incapazes de desenvolver o programa devido a dificuldades exclusivamente ligadas à programação, dos alunos com dificuldades de cariz matemático, relacionadas com o conceito em questão.

Na questão 3 foi pedido aos alunos que desenvolvessem um programa que calculasse a soma de duas fracções:

> *3 - Imagine que pretende fazer um programa para calcular a soma de duas fracções.*
>
> *a) Transforme o programa anterior numa função que receba dois números e calcule e devolva o m.m.c. entre eles.*
>
> *b) Faça o algoritmo do programa que calcula a soma de duas fracções, usando a função anterior. Devem ser introduzidos os 4 valores correspondentes às fracções N1/D1 e N2/D2 e mostrado o resultado da soma no formato N3/D3.*

Neste exercício era pedido, primeiramente, a transformação do programa criado na questão 2 numa função que recebesse dois números e devolvesse o m.m.c. associado. Posteriormente, era pedido aos alunos que usassem essa função num programa que calculasse a soma de duas fracções usando o m.m.c. dos seus denominadores. Através deste procedimento, conseguimos identificar os alunos que mostraram capacidade para resolver

problemas de programação e que poderiam seguir um raciocínio lógico e consistente, de forma a permitir-lhes desenvolver outros níveis e áreas de programação.

Os alunos que demonstraram dificuldades com o conceito de m.m.c. (1ª questão do 1º teste) foram direccionados para um outro teste (Teste1.2), focalizado em aspectos matemáticos sobre esse conceito. Os que demonstraram dificuldades apenas do ponto de vista da programação, na questão 2 foram direccionados para outro teste de programação (Teste1.3), a fim de podermos identificar detalhadamente este tipo de dificuldades. Os alunos que não demonstrassem dificuldades de nenhum tipo, iam avançando no sentido de resolver os problemas de programação de dificuldade crescente do Teste 1.1.

No teste (Teste 1.2), era fornecida uma explicação teórica do conceito de m.m.c., complementada com explicações usando exemplos concretos, de forma a que os alunos pudessem apreender o conceito na sua plenitude. De seguida, os alunos foram confrontados com um conjunto de quatro questões com um nível crescente de dificuldade. Desta forma pudemos captar as dificuldades dos alunos, desde um nível muito básico (conhecimento do conceito de m.m.c.) – através da sua aplicação directa - a exercícios onde os alunos deveriam tecer as suas próprias conclusões, a partir de níveis anteriores, generalizando-as e adaptando-as a novas situações.

Assim, na questão 1 era pedido aos alunos que indicassem, numa lista de números, qual o m.m.c. entre dois números dados. Esta questão ajudou a identificar os alunos que não distinguiam os conceitos de m.m.c. e divisores ou múltiplos de números.

A questão 2 do mesmo teste pedia aos alunos que calculassem o m.m.c. entre três números. A ideia era a de identificar os alunos que compreendiam correctamente o conceito de m.m.c. entre dois ou mais números e que, subsequentemente, conseguiam fazer alguma generalização. De seguida, na questão 3, aos alunos era solicitado que resolvessem um problema mais alargado envolvendo, implicitamente, o conceito de m.m.c.

> *3 - Dois amigos encontraram-se na cidade onde nasceram em Dezembro de 2007. Um deles regressa de 3 em 3 meses e o outro de 4 em 4 meses. Quando é que os amigos se voltaram a encontrar, na cidade considerada?*

Esta questão envolvia a análise e compreensão de determinado problema requerendo uma determinada abordagem matemática para a sua resolução. Finalmente, a questão 4 foi construída com base na anterior, envolvendo nova generalização do conceito de m.m.c.

> *4 - O que aconteceria se existisse um terceiro amigo que se encontrasse, na cidade, no mesmo dia dos outros dois e que retornasse à cidade de 5 em 5 meses? Isto é, quando é que os três se voltariam a encontrar?*

O 3º teste (Teste 1.3), de programação, incluía um conjunto de questões de dificuldade crescente, todas inter-relacionadas com o conceito matemático m.m.c. A ideia geral era a de verificar o nível de compreensão do aluno sobre conceitos matemáticos básicos e, para os alunos que demonstrassem uma plena compreensão, verificar até que ponto conseguiam fazer um programa envolvendo os mesmos conceitos. As perguntas foram escolhidas de modo a reflectir os diferentes níveis da taxonomia de Bloom, neste caso aplicado aos aspectos de programação. O objectivo era o de identificar as dificuldades dos alunos, bem como o nível em que estas começavam a surgir.

Assim, na questão 1 os alunos receberam algumas linhas de código

```
1: var n1, n2, i, j :integer;
...
2: n1 := 3;
3: n2 := 10;
4: for i:=n1 to n2 do
5:   if i mod 2 = 0
6:     then write(i);
```

que tinham como finalidade imprimir os números pares entre 3 e 10.

A questão incluía sete alíneas muito básicas, ao nível do Conhecimento, apenas para verificar se os alunos conseguiam reconhecer e compreender o efeito de conceitos básicos de programação, a saber, variáveis; instruções da atribuição, instruções de entrada/saída e estruturas de controlo básicas (selecção e repetição).

1a) Identifique as variáveis.

1b) Qual o valor inicial de cada uma das variáveis? Indique o número da linha de código onde a variável é iniciada.

1c) Supondo i=57 qual o resultado da seguinte expressão i:=i+1?

1d) Supondo i=101 qual o resultado da seguinte expressão i mod 2?

1e) Existe algum ciclo? Se sim, indique qual a linha em que é iniciado e terminado?

1f) Existe alguma instrução de entrada e/ou saída de dados? Diga quais e em que linha(s)?

1g) Explique o que faz a instrução for i:=n1 to n2 do.

As questões 2 e 3 situavam-se ao nível da Compreensão. A primeira pretendia verificar se os alunos conseguiam interpretar a totalidade do código apresentado na questão 1, bem como qual a sua saída. A questão 3 verificava se o aluno compreendia a importância da ordem pela qual os valores eram atribuídos às variáveis e de como essa atribuição afectava o

programa. Os alunos deveriam aperceber-se da ocorrência de uma situação não desejada quando a troca dos valores das variáveis era realizada. Era suposto que os alunos se conseguissem expressar relativamente ao porquê de usar determinada condição, bem como ao porquê de determinada condição não poder ser substituída por outra.

2 - Explique, de forma sintetizada, o que faz o pedaço de código acima.

3 - Assuma que as linhas 2 e 3 são substituídas pelo seguinte código:

```
2: n1:=10;
3: n2:=3;
```

Qual o resultado desta modificação no código acima?

A questão 4, situada ao nível de Aplicação, pedia que os estudantes alterassem o código de forma a obter um determinado comportamento.

4 - Introduza as alterações necessárias no programa dado (em 1), considerando os dados da pergunta anterior, de forma a que o comportamento do programa seja o mesmo do que o dado (em 1).

Os alunos que conseguiram responder a este problema correctamente demonstraram compreender o código na sua totalidade, dominando técnicas de programação básicas e revelando a capacidade de raciocínio que lhes permitia a remodelação de programas de acordo com novas especificações.

A questão 5 pretendia averiguar se os alunos conseguiam extrair uma abstracção, indicando o objectivo geral de uma instrução e não apenas o significado literal dessa instrução pelo que a considerámos ao nível da Análise.

5 - Explique a razão da condição if i mod 2 = 0?

Considerámos que as questões 6, 7 e 8 estavam ao nível de Síntese. Nessas perguntas era pedido aos alunos para fazerem umas mudanças mais elaboradas no programa, a fim de obter uma nova saída. Nestas questões os alunos deveriam relacionar os conceitos matemáticos e de programação usados até esse momento e aplicá-los para imprimir os múltiplos de um número dado, pertencentes a determinado intervalo (pergunta 6), imprimir os múltiplos comuns de dois números pertencentes a determinado intervalo (pergunta 7) e finalmente (pergunta 8) imprimir o m.m.c. de dois números dados. De notar que todas estas perguntas estavam relacionadas e exigiam uma completa compreensão relativa aos conceitos matemáticos inerentes ao m.m.c., bem como aos conceitos fundamentais relativos a programação, treinados em níveis precedentes.

> *6 - Com base no programa dado, escreva um programa que imprima os múltiplos de n1 existentes entre n1 e n1*n2.*
>
> *7 - Evolua o programa anterior, de modo a imprimir os múltiplos comuns de n1 e n2, pertencentes ao intervalo entre n1 e n1*n2.*
>
> *8 - Altere o programa anterior de forma a que seja apenas mostrado o mínimo múltiplo comum entre n1 e n2.*

A última questão, 9,

> *9 - Será necessário o ciclo percorrer todos os números intermédios entre n1 e n2? Que alterações sugere para optimizar o procedimento de cálculo do mínimo múltiplo comum entre n1 e n2.*

foi enquadrada no último nível da taxonomia de Bloom (Avaliação). Nesta questão era pedido aos alunos que alterassem o programa que determinava o m.m.c. entre dois números, a fim de aperfeiçoá-lo, evitando operações desnecessárias. Da nossa perspectiva, os alunos que conseguissem resolver este problema tinham interiorizado o conceito matemático de m.m.c. bem como os conceitos básicos de programação exigidos para desenvolver os algoritmos relativos ao m.m.c.

Dos testes distribuídos na 2ª sessão, um tinha perguntas exclusivamente sobre conceitos matemáticos simples (Anexo C.4 – Teste 2.1) e o outro (Anexo C.5 – Teste 2.2) tinha perguntas de programação incluindo conceitos envolvidos no primeiro. No 1º teste, um dos objectivos era perceber se os alunos conseguiam relacionar os conceitos de múltiplos e divisores de um número. Se tal não acontecesse, era porque tinham resolvido as questões da sessão anterior por mecanização ou listagem de múltiplos, não tendo apreendido o conceito na sua essência. As diversas perguntas tinham por objectivo, perceber se os alunos compreendiam a decomposição de um número em factores primos; se conseguiam estabelecer relações entre a factorização em números primos e o conceito de múltiplo e se os alunos estavam capacitados para se aperceberem, sozinhos, de um método simples de obtenção do m.m.c entre dois ou mais números. Em qualquer dos testes tentou-se que as questões seguissem aproximadamente a taxonomia de Bloom. Adicionalmente, foi fornecida uma folha com várias definições dos conceitos matemáticos necessários a cada um dos testes, exemplos que ilustravam esses conceitos, bem como descrições algorítmicas sobre os métodos para obter esses procedimentos.

6.4.4 Análise de resultados

Face ao tipo de informação que se pretendia extrair utilizou-se um estudo observacional, com uma análise qualitativa descritiva. Assim, relativamente às competências de resolução de problemas com ênfase matemática, de suporte ao factor F1.2, pudemos constatar as dificuldades de seguida apresentadas.

Na 1ª sessão dois alunos não responderam à questão relativa ao m.m.c. entre 2 números, pelo que concluímos que não sabiam o conceito. Outros dois alunos responderam erradamente, (21=3x7;6=2x3, então m.m.c é 3), demonstrando saber a decomposição de um número em factores primos, mas não conseguindo alcançar a resposta correcta. Três alunos responderam correctamente à pergunta, mas dois só o conseguiram fazer por tentativa e erro, listando os múltiplos até encontrarem um resultado comum. Estes 3 alunos que acertaram a resposta referente ao m.m.c. foram aconselhados a fazer o programa para calcular o m.m.c. entre 2 números (pergunta 2 do Teste 1.1).

1 - Determine o m.m.c. entre 6 e 21.

	Nº de alunos
Respostas Correctas	3
Respostas Incorrectas	2
Não Responderam	2

Tabela 6.15: Resposta à Questão 1 do Teste 1.1

Destes 3 alunos, 2 começaram a tentar responder à questão, mas apenas escreveram cabeçalhos, declaração de variáveis e instruções para pedir e ler valores. O outro aluno tentou fazer o programa, mas não viu a necessidade de um ciclo, tentou resolver o problema com uma selecção e a codificação apresentada não fazia o mínimo sentido. Este aluno, ainda prosseguiu para a pergunta seguinte mas como apenas escreveu cabeçalhos, declaração de variáveis e instruções para pedir e ler valores, foi direccionado para o Teste 1.3.

2 - Escreva um programa que calcule o m.m.c. entre dois números.

	Nº de alunos
Respostas Correctas	0
Respostas Incorrectas	3

Tabela 6.16: Resposta à Questão 2 do Teste 1.1

Dos quatro alunos que aparentaram não conhecer o conceito de m.m.c. e que foram redireccionados para o Teste 1.2 (sobre conceitos matemáticos) todos conseguiram resolver os três primeiros exercícios.

1 - Identifique a resposta correcta, o m.m.c. entre 7 e 21 é:

a) 3

b) 147

c) 21

2 - Determine o m.m.c (12,20,24).

3 - Dois amigos encontraram-se na cidade onde nasceram em Dezembro de 2007. Um deles regressa de 3 em 3 meses e o outro de 4 em 4 meses. Quando é que os amigos se voltaram a encontrar, na cidade considerada?

	Nº de alunos
Respostas Correctas	4
Respostas Incorrectas	0

Tabela 6.17: Resposta às Questões 1, 2 e 3 do Teste 1.2

No entanto, apenas 2 alunos acertaram o último exercício que envolvia uma transferência do conceito para uma situação prática generalizada a três números. O que achámos curioso, pois estávamos à espera que aqueles alunos que conseguiram responder à questão 3 também o fizessem para a questão 4, pois o tipo de raciocínio era o mesmo, apenas envolvia mais um número.

4 - O que aconteceria se existisse um terceiro amigo que se encontrasse, na cidade, no mesmo dia dos outros dois e que retornasse à cidade de 5 em 5 meses? Isto é, quando é que os três se voltariam a encontrar?

	Nº de alunos
Respostas Correctas	2
Respostas Incorrectas	2

Tabela 6.18: Resposta à Questão 4 do Teste 1.2

Apesar de em momentos diferentes, todos os sete alunos foram direccionados para o outro teste exclusivo de programação (Teste 1.3).

1 - Dado o seguinte extracto de código

```
1: var n1, n2, i, j :integer;
...
2: n1 := 3;
3: n2 := 10;
4: for i:=n1 to n2 do
5:       if i mod 2 = 0
6:             then write(i);
```

Neste, relativamente à questão 1a), quatro alunos identificaram todas as variáveis e três identificaram apenas duas variáveis (n1 e n2).

1a) Identifique as variáveis.

	Nº de alunos
Respostas Correctas	4
Respostas Incorrectas	0
Respostas Semi-correctas	3

Tabela 6.19: Resposta à Questão 1a) do Teste 1.3

Relativamente à questão 1b), um aluno conseguiu descobrir os valores iniciais apenas das variáveis n1 e n2, mas não especificou as suas linhas de iniciação e quatro alunos conseguiram descobrir os valores iniciais apenas das variáveis n1 e n2, especificando as suas linhas de iniciação, mas não consideraram i e j como variáveis (um aluno, em conversa com a investigadora, fez ainda a seguinte afirmação "se j tem sempre o mesmo valor, não é iniciado, logo é uma constante e não uma variável"). Um aluno conseguiu descobrir os valores iniciais de n1 e n2, especificando correctamente as suas linhas de iniciação mas não o conseguiu fazer para a variável i. Apenas 1 aluno conseguiu descobrir os valores iniciais de n1, n2 e i (referindo neste caso ser igual a n1, mas não concretizando qual o valor), referindo que j não era iniciado e especificando as linhas de iniciação correctas.

1b) Qual o valor inicial de cada uma das variáveis? Indique o número da linha de código onde a variável é iniciada.

	Nº de alunos
Respostas Correctas	1
Respostas Incorrectas	0
Respostas Semi-correctas	6

Tabela 6.20: Resposta à Questão 1b) do Teste 1.3

Todos os alunos responderam correctamente às perguntas 1c) e 1d), o que demonstra o conhecimento da sintaxe de uma linguagem de programação e a capacidade de perceber instruções concretas e isoladas.

1c) Supondo i=57 qual o resultado da seguinte expressão i:=i+1?

1d) Supondo i=101 qual o resultado da seguinte expressão i mod 2?

	Nº de alunos
Respostas Correctas	7
Respostas Incorrectas	0

Tabela 6.21: Resposta às Questões 1c) e 1d) do Teste 1.3

Relativamente à questão 1e), um aluno referiu a não existência de qualquer ciclo, três alunos responderam acertadamente, revelando saber onde o ciclo começava e acabava e dois alunos referiram a existência de um ciclo, mas desconhecendo ou errando a linha onde acabava. Um aluno reconheceu a existência de um ciclo, mas não soube identificar onde começava ou terminava (pensamos que este aluno não percebeu a pergunta, pois reportou a resposta aos valores limites).

1e) Existe algum ciclo? Se sim, indique qual a linha em que é iniciado e terminado?

	Nº de alunos
Respostas Correctas	3
Respostas Incorrectas	1
Respostas Semi-correctas	3

Tabela 6.22: Resposta à Questão 1e) do Teste 1.3

Relativamente à questão 1f), dois alunos referiram não existir qualquer instrução de entrada/saída, um aluno referiu erradamente a existência de uma instrução de entrada (considerou a declaração de variáveis como entrada de dados) e acertou na linha correspondente à instrução de saída. Dois alunos identificaram erradamente a instrução de entrada de dados e correctamente a de saída. Um aluno identificou apenas correctamente a instrução de saída. Apenas um aluno respondeu correctamente à totalidade da pergunta.

1f) Existe alguma instrução de entrada e/ou saída de dados? Diga quais e em que linha(s)?

	Nº de alunos
Respostas Correctas	1
Respostas Incorrectas	2
Respostas Semi-correctas	4

Tabela 6.23: Resposta à Questão 1f) do Teste 1.3

Relativamente à questão 1g), quatro alunos pareceram ter percebido o comportamento da instrução pedida, pois responderam correctamente, dois alunos responderam erradamente, referindo que o valor i iria apenas tomar o valor 3 e/ou o valor 10. Um aluno não respondeu.

1g) Explique o que faz a instrução for i:=n1 to n2 do.

	Nº de alunos
Respostas Correctas	4
Respostas Incorrectas	2
Não Responderam	1

Tabela 6.24: Resposta à Questão 1g) do Teste 1.3

Relativamente à questão 2, dois alunos responderam correctamente, um aluno conseguiu apenas detectar o teste de múltiplo de 2 mas não conseguiu dar uma resposta completa. Dois alunos conseguiram ver a ideia do ciclo e quais os valores que eram percorridos mas não o teste de divisibilidade por 2 (teste de paridade). Um aluno não respondeu e outro aluno deu uma resposta completamente errada.

2 - Explique, de forma sintetizada, o que faz o pedaço de código acima.

	Nº de alunos
Respostas Correctas	2
Respostas Incorrectas	1
Respostas Semi-correctas	3
Não Responderam	1

Tabela 6.25: Resposta à Questão 2 do Teste 1.3

Quanto à questão 3, quatro alunos responderam erradamente (um respondeu que a consequência era a variável passar a ser ímpar…), dois alunos não responderam e apenas um aluno acertou na totalidade.

3 - Assuma que as linhas 2 e 3 são substituídas pelo seguinte código:

```
2: n1:=10;
3: n2:=3;
```

Qual o resultado desta modificação no código acima?

	Nº de alunos
Respostas Correctas	1
Respostas Incorrectas	4
Não Responderam	2

Tabela 6.26: Resposta à Questão 3 do Teste 1.3

Relativamente à questão 4, um aluno deu uma resposta errada, outro aluno acertou completamente e cinco alunos deram uma resposta que não se encontrava completamente certa, mas que para os objectivos da pergunta pensamos poder considerar-se como certa. Em diálogo com estes alunos, verificámos a dificuldade que tiveram em perceber o enunciado. De notar que a própria investigadora revelou alguma dificuldade para conseguir encontrar um texto que expressasse as suas intenções.

4 - Introduza as alterações necessárias no programa dado (em 1), considerando os dados da pergunta anterior, de forma a que o comportamento do programa seja o mesmo que o dado (em 1).

	Nº de alunos
Respostas Correctas	6
Respostas Incorrectas	1

Tabela 6.27: Resposta à Questão 4 do Teste 1.3

Relativamente à questão 5, quatro alunos acertaram e três erraram. De notar a diferença de resultados entre esta questão e a questão 1d). Embora não sendo significativo, na questão 1d) todos os alunos conheciam o significado concreto da mesma instrução utilizada. Os resultados da questão 5 evidenciam a incapacidade de generalização de alguns alunos, mesmo perante perguntas extremamente simples.

5 - Explique a razão da condição if i mod 2 = 0?

	Nº de alunos
Respostas Correctas	4
Respostas Incorrectas	3

Tabela 6.28: Resposta à Questão 5 do Teste 1.3

No que respeita à pergunta 6, três alunos acertaram, um errou (indicando o limite superior como n2 em vez de n1*n2 e estabelecendo a condição de teste errada) e três alunos não responderam.

*6 - Com base no programa dado, escreva um programa que imprima os múltiplos de n1 existentes entre n1 e n1*n2.*

	Nº de alunos
Respostas Correctas	3
Respostas Incorrectas	1
Não Responderam	3

Tabela 6.29: Resposta à Questão 6 do Teste 1.3

Em termos da pergunta 7, dois alunos acertaram, quatro não responderam e um errou.

*7 - Evolua o programa anterior, de modo a imprimir os múltiplos comuns de n1 e n2, pertencentes ao intervalo entre n1 e n1*n2.*

	Nº de alunos
Respostas Correctas	2
Respostas Incorrectas	1
Não Responderam	4

Tabela 6.30: Resposta à Questão 7 do Teste 1.3

Relativamente à pergunta 8, um aluno respondeu erradamente e seis não responderam.

> *8 - Altere o programa anterior de forma a que seja apenas mostrado o mínimo múltiplo comum entre n1 e n2.*

	Nº de alunos
Respostas Correctas	0
Respostas Incorrectas	1
Não Responderam	6

Tabela 6.31: Resposta à Questão 8 do Teste 1.3

A pergunta 9 foi respondida correctamente na sua essência, mas não na totalidade, por um aluno, havendo seis alunos que não responderam.

> *9 - Será necessário o ciclo percorrer todos os números intermédios entre n1 e n2? Que alterações sugere para optimizar o procedimento de cálculo do mínimo múltiplo comum entre n1 e n2.*

	Nº de alunos
Respostas Correctas	1
Respostas Incorrectas	0
Não Responderam	6

Tabela 6.32: Resposta às Questão 9 do Teste 1.3

Estes resultados evidenciam determinados aspectos referentes às dificuldades dos alunos a programação, nomeadamente o facto de a maioria dos alunos conseguir interpretar instruções simples e isoladas, apresentar dificuldades em generalizar comportamentos de instruções, ter mais dificuldades em escrever pequenos pedaços de código mesmo que em contextos bem definidos e revelar-se incapaz de escrever programas completos, mesmo que similares a outros apresentados.

Na 2ª sessão, houve um decréscimo do número de alunos voluntários, ficando a amostra reduzida a três alunos. Face a este cenário, tentámos obter perfis de conhecimentos referentes ao conjunto das duas sessões que passamos a descrever.

Um aluno demonstrou saber o conceito de m.m.c. entre dois números, mas apenas o conseguiu determinar listando sucessivamente os múltiplos comuns aos números. Este aluno

também demonstrou saber o conceito de número primo, bem como obter a factorização de um número em números primos, mas não conseguiu obter o m.m.c. através do algoritmo da factorização, nem responder a questões que relacionassem estes dois conceitos. Em termos de programação, este aluno demonstrou saber as instruções básicas (iniciação, entrada/saída, estruturas de selecção/decisão,..), no entanto, para o cálculo do m.m.c. recorreu ao conceito de máximo divisor comum (m.d.c.) que não soube calcular. Ao longo dos exercícios demonstrou reconhecer instruções e operações básicas, a iniciação explícita de variáveis, não reconhecendo, no entanto, como variáveis aquelas cuja iniciação não era feita de forma explícita. Estas falhas foram reconhecidas nas duas sessões. Este aluno revelou não ter um verdadeiro entendimento do conceito de ciclo, considerando numa sessão a estrutura repetitiva (`for`) como um ciclo e na seguinte, além dessa, também considerou como ciclo a estrutura de selecção (`if`). Apesar de conhecer e compreender o conceito de número primo do ponto de vista matemático este aluno não conseguiu desenvolver um programa que implementasse esse conceito. Neste caso, limitou-se às instruções de iniciação e entrada/saída, não fazendo qualquer tentativa para obter o processo de cálculo de números primos. Este aluno pareceu compreender a estrutura repetitiva (`for`), por análise e aplicação directa, nas diversas situações em que houve necessidade. Nalgumas situações demonstrou uma compreensão do significado das instruções, não se limitando à sua simples tradução (por exemplo, fazendo o seguinte tipo de afirmações `if i mod 2=0`, testa se i é múltiplo de 2). Este aluno também conseguiu fazer um programa para imprimir os múltiplos de um número num dado intervalo, mas já não conseguiu fazer um programa ligeiramente diferente para imprimir todos os múltiplos comuns a dois números dados, pertencentes a determinado intervalo. Nesta última situação fez uma tentativa, recorrendo novamente ao m.d.c. que não tinha bem alicerçado. Apesar da sequência de exercícios tencionar encaminhar o aluno para uma resolução correcta, o último exercício realizado por este aluno (questão 8 do Teste 1.3) consistiu num programa exactamente igual ao pedido na questão 6. Este aluno, na 2ª sessão, face ao programa fornecido, conseguiu identificar todas as instruções, pedaços de código isolados e o programa na sua totalidade. Apesar de nalgumas situações em que eram pedidas explicações acerca de comportamentos de instruções isoladas ou em conjunto, se deduza que o aluno muitas vezes as compreendeu, na realidade não se conseguia exprimir convenientemente. No entanto, quando lhe foi pedida a obtenção de um programa (cálculo dos divisores comuns entre dois números) ligeiramente diferente do apresentado o aluno revelou-se incapaz de o fazer. O aluno fez uma tentativa de implementação, mas em vez de calcular os divisores comuns entre dois números, verificou se um número era divisível por outro. Na pergunta 3 do Teste2.2 quando se pedia para completar um programa ao qual faltava a condição lógica para terminar a estrutura repetitiva `REPEAT…UNTIL`, tendo o aluno que escolher uma opção de entre quatro, o aluno

escolheu a opção correcta mas não justificou, pelo que acabámos por não perceber se compreendeu quer a estrutura quer as condições lógicas que a compunham. No entanto, relativamente às condições lógicas demonstrou o seu entendimento noutra pergunta anterior. Concluindo, este aluno demonstrou saber interpretar pedaços de código e programas completos e apesar de conseguir proceder a alterações simples a um programa dado, não conseguiu construir um programa de forma integral, mesmo que similar ao fornecido.

Outro aluno demonstrou não saber o conceito de m.m.c. entre dois números. Quando submetido à ficha explicativa do conceito, conseguiu responder a questões directas sobre o m.m.c. entre dois ou mais números, apesar de já não conseguir aplicar o conceito numa situação concreta envolvendo mais de dois números. Na 2ª sessão este aluno demonstrou saber outros conceitos matemáticos como a factorização em números primos. Em termos de programação, demonstrou saber as instruções básicas (iniciação, estruturas de selecção/decisão,..), identificando todas as variáveis. No entanto, quanto às instruções de entrada/saída, identificou a instrução de saída correctamente, mas considerou como instrução de entrada os limites do ciclo `for`. Este tipo de deficiências foi demonstrado nas duas sessões. A análise de diversas codificações feitas por este aluno revela que não compreende verdadeiramente o comportamento de um ciclo `for`. Por exemplo, quando questionado sobre o que acontecia quando o 1º limite do ciclo apresentava um valor superior ao 2º, o aluno considerou que a variável de controlo assumia apenas estes dois valores. O aluno não se apercebeu da impossibilidade, nesta situação, de o ciclo ser iniciado. Na 1ª sessão o aluno não conseguiu fazer qualquer programa pedido, uma vez que, as tentativas que realizou não faziam qualquer sentido. Em geral, este aluno, conseguiu traduzir literalmente um pedaço de código, mas não conseguiu abstrair a sua essência. Não conseguia explicitar a real utilidade de um pedaço de código ou de um programa na sua totalidade. Em diversas situações, reportou as suas respostas aos dados concretos do programa, limitando-se a traduzir as instruções literalmente. No entanto, na 2ª sessão, quando lhe foi pedido a obtenção de um programa (cálculo dos divisores comuns entre 2 números) ligeiramente diferente do fornecido (cálculo do m.m.c. entre dois números) o aluno conseguiu fazê-lo, embora não da maneira mais eficiente. Conclui-se assim que este aluno conseguiu traduzir pedaços de código mas não os conseguiu interpretar ou generalizar, o mesmo acontecendo com um programa integral. Surpreendentemente, conseguiu implementar correctamente um programa similar ao fornecido, ainda que não da forma mais eficiente.

Outro aluno, apesar de ter acertado a pergunta referente ao m.m.c., a sua forma de resolução pareceu indicar que não tinha o conceito bem apreendido, pelo que foi redireccionado para a ficha exemplificativa deste conceito, que resolveu integral e

correctamente. Parecia saber o conceito de número primo, bem como obter a factorização de um número em números primos, se bem que no cálculo de uma das factorizações tenha cometido um erro que pareceu de distracção. Não conseguiu obter o m.m.c. através do algoritmo da factorização, nem responder a questões que relacionassem estes dois conceitos. Em termos de programação, este aluno não conseguiu identificar as instruções de entrada e saída de dados (`read e write`), indicando no seu lugar as palavras reservadas `INPUT`, `OUTPUT`. Este tipo de respostas parece indiciar que certas designações sintácticas em determinadas linguagens de programação, só por si, poderão ser fontes causadoras de dificuldades. Ao longo dos exercícios, este aluno demonstrou reconhecer instruções e operações básicas, a iniciação explícita de variáveis, não reconhecendo, no entanto, como variáveis aquelas cuja iniciação não era feita de forma explícita. Este aluno revelou não ter um verdadeiro entendimento do conceito de ciclo, considerando numa sessão a estrutura repetitiva (`for`) como um ciclo e na seguinte, além dessa também considerou como ciclo a estrutura de selecção (`if`). De notar que esta confusão também ocorreu com outro aluno. Relativamente à estrutura repetitiva, identificou correctamente onde começava e terminava o ciclo, mas pareceu não revelar um completo entendimento do seu comportamento. Conseguiu traduzir as instruções de um pedaço de código literalmente, mas pareceu não compreender a sua real utilidade, o mesmo acontecendo com a totalidade de um programa. Reportou as suas respostas a dados concretos do programa, não conseguindo ou pelo menos não indicando nas suas respostas generalizações de determinados comportamentos. Quando lhe foi pedida a obtenção de um programa (cálculo dos divisores comuns entre dois números) ligeiramente diferente do fornecido (cálculo do m.m.c. entre dois números) o aluno não conseguiu fazê-lo, limitando-se à introdução de instruções de leitura e escrita e tentando explicar por palavras, o algoritmo para a sua resolução, embora erradamente. Na pergunta em que era pedido para completar um programa ao qual faltava a condição lógica para terminar a estrutura repetitiva `REPEAT…UNTIL`, através da escolha de uma opção de entre quatro opções, o aluno escolheu a opção correcta. Porém, a sua justificação consistiu na tradução literal da instrução, pelo que acabámos por não perceber se compreendeu a utilidade desta instrução. Este aluno pareceu não perceber a utilidade da instrução `break`.

6.4.5 Conclusões

Este estudo embora realizado com alunos de um curso de Matemática, que teoricamente teriam melhores conhecimentos matemáticos e capacidades de resolução de problemas quando comparado com o nosso público-alvo principal (alunos de Engenharia Informática), permitiu-nos recolher um conjunto de dificuldades apresentadas na tabela 6.33.

Embora de uma forma qualitativa e interpretativa, pensamos que os resultados apresentados nos fornecem indícios relativamente às nossas convicções sobre o factor F2.3 – As tarefas habitualmente propostas pelos professores apresentam níveis de dificuldades desajustados ao nível cognitivo do aluno. Isto porque, o tipo de tarefas usadas neste estudo, mesmo que usando uma estratégia que julgamos mais adequada relativamente às tradicionalmente utilizadas, ainda assim se revelou desajustada. Apesar dos alunos utilizados nesta amostra serem alunos muito fracos, conseguindo atingir apenas o nível de Conhecimento ou Compreensão, provou-se que a maioria das actividades propostas estava muito além das suas capacidades cognitivas. Apesar de desconhecermos o tipo de actividades normalmente realizadas pela professora da disciplina e de eventualmente existirem alunos com capacidades superiores, conversas informais levam-nos também a reforçar estas convicções.

Adicionalmente, esta experiência, permitiu reforçar a convicção relativa ao factor F1.2 – "Muitos alunos com dificuldades de aprendizagem de programação apresentam défices de conhecimentos matemáticos e lógicos", na medida em que verificámos que os alunos com dificuldades a programação apresentam também enormes dificuldades em resolver problemas de base matemática e em especial no raciocínio lógico que possibilita a sua generalização. O curioso foi verificar que este não é um problema específico das Engenharias mas também comum a alunos de outras licenciaturas como a de Matemática.

Estudo C - Conclusões	
Dificuldades matemáticas	Os alunos não dominavam muitos conceitos matemáticos básicos. Apesar de os alunos, por vezes, resolverem problemas envolvendo certos conceitos matemáticos não os compreendiam verdadeiramente ou não os sabiam aplicar. Os alunos tinham fracos níveis de abstracção.
Dificuldades de programação	De uma forma geral, os alunos não demonstraram, pelo menos em termos escritos, grande esforço em tentativas de solução, traduzidas através de codificações. A maioria dos erros cometidos pelos alunos eram do foro lógico e não erros sintácticos. Nas poucas codificações que fizeram, os alunos revelaram saber a sintaxe de algumas instruções, apresentando maiores dificuldades na compreensão dos ciclos e variáveis. A generalidade dos alunos conseguiu identificar correctamente as variáveis, mas nem sempre conseguiu identificar os seus valores iniciais. Relativamente às estruturas repetitivas, somente alguns as reconheceram e souberam indicar a linha onde começavam e terminavam. A finalidade do ciclo era, por vezes, desconhecida ou não completamente compreendida por muitos alunos. Nem todos os alunos reconheceram as instruções de entrada e/ou saída. Somente alguns alunos compreenderam conjuntos interligados de instruções. As dificuldades aumentaram quando aos alunos era pedido para explicarem a funcionalidade da totalidade de um programa ou a sua saída em consequência de determinadas alterações de código. A partir do momento em que eram pedidas alterações de código, mesmo que em contextos bem delimitados, os alunos começaram a desistir ou a dar respostas erradas. As dificuldades em conseguir juntar as partes de código ou produzir codificações foram as mais evidentes, revelando aspectos cognitivos mais exigentes para os quais os alunos não estavam preparados. A maioria dos alunos conseguiu traduzir pedaços de código isolados não os conseguindo, muitas vezes, interpretar ou generalizar, em especial quando esse código era constituído por várias instruções, e piorando com programas completos. A maioria dos alunos não conseguiu implementar o núcleo de qualquer programa, mesmo que similar aos fornecidos.

Tabela 6.33: Conclusões do Estudo C

6.5 Estudo D

Este estudo (Gomes e Mendes, 2009), não foi planeado inicialmente, mas a ocasião de ser responsável da disciplina de Tecnologia da Informática, deu oportunidade de estudar novamente o factor F2.1 – A Engenharia Informática/Ciências da Computação atrai alunos com determinado perfil de estilos de aprendizagem. Porém, o principal objectivo deste estudo foi o de testar o factor F2.3 – "As tarefas habitualmente propostas pelos professores apresentam níveis de dificuldades desajustados ao nível cognitivo do aluno", bem como a eficácia da utilização de uma taxonomia para melhor diagnosticar as dificuldades dos alunos. A escolha da disciplina referida para realizar o estudo teve essencialmente dois motivos. Por um lado, todo a curso de Engenharia Informática foi completamente reformulado no ano lectivo em que o estudo decorreu, em consequência da aplicação das deliberações da reforma de Bolonha. Esta reformulação teve necessariamente em consideração reflexões sobre metodologias de ensino, aprendizagem e avaliação. A conjugação desta situação com a autonomia que poderíamos exercer sobre esta disciplina possibilitou a criação de um conjunto de actividades lectivas e materiais de avaliação, de acordo com uma taxonomia de objectivos educacionais. Por outro lado, a razão para a escolha desta disciplina para a realização do estudo prende-se com o facto de, na componente prática se estudar programação em Assembly, linguagem sobre a qual não é comum encontrar estudos relativamente à problemática estudada.

6.5.1 Contexto

Este estudo decorreu no Departamento de Engenharia Informática e Sistemas do Instituto Superior de Engenharia de Coimbra (DEIS-ISEC), no 4º trimestre do ano lectivo de 2007/2008, envolvendo alunos inscritos no curso de Licenciatura em Engenharia Informática (LEI). A experiência foi baseada nos resultados obtidos à disciplina de Tecnologia da Informática que inclui, na sua componente prática, a aprendizagem de programação básica em Assembly.

6.5.2 Instrumentos e Metodologia

Neste estudo o principal instrumento, que permitiu recolher e analisar os dados de forma significativa foi o exame final da disciplina (Anexo D). Adicionalmente foi utilizado o *Index of Learning Styles* (Anexo G), em contextos similares aos já referidos neste documento, noutros estudos que o utilizaram.

Em termos metodológicos foram usados conjuntos de procedimentos que na nossa óptica condicionaram alguns aspectos referentes aos resultados obtidos no exame final. Em

simultâneo com a reforma de Bolonha ocorreu uma mudança em termos de organização temporal das disciplinas, passando de um regime semestral para um regime trimestral. Com esta nova estrutura o número de disciplinas que cada aluno tinha simultaneamente era metade do que anteriormente, cada uma das quais passando a ter uma carga horária duplicada. Esta nova aproximação implicou que os alunos tivessem, semanalmente, muitas horas do contacto com o mesmo professor. Este aspecto, fez com que fosse ainda mais importante usar uma estratégia inovadora e diversificada, que incluísse diferentes tipos de actividades e mantivesse um ambiente de sala de aula mais informal, de forma a evitar a saturação dos alunos. Desta forma, incluímos actividades diversificadas após cada período de ensino tradicional, basicamente seguindo a taxonomia dos objectivos educacionais de Bloom. Estas tarefas, realizadas em contexto de sala de aula, eram distribuídas aos alunos que as realizavam em grupos ou de forma isolada, consoante as preferências, mas sob a supervisão e o apoio do professor. Sempre que o professor considerava oportuno havia momentos para discussão das soluções e explicação das resoluções. Considerando a estratégia pedagógica seguida durante as aulas, decidiu-se também estruturar a componente prática do exame final de acordo com a taxonomia de Bloom. Isto é, em vez de a componente prática do exame consistir apenas em perguntas em que é pedida a realização de programas completos (como era usual anteriormente à reformulação) estruturou-se o exame em três grupos. Todos os grupos incluíram questões com um grau crescente de dificuldade estruturadas com base na taxonomia de Bloom.

A primeira pergunta incluía um grupo de alíneas estruturadas em quatro níveis de dificuldade. O primeiro nível incluiu quatro alíneas nas quais os alunos tinham de identificar, no programa fornecido, o local onde as variáveis eram declaradas, o espaço e endereços ocupados por algumas dessas variáveis, bem como indicar a existência ou não de um ciclo, em determinado fragmento de código. Estas questões testaram o nível de Conhecimento da taxonomia de Bloom. O segundo nível de questões incluiu alíneas para testar se os alunos conseguiam identificar não apenas algumas instruções, mas se igualmente compreendiam o seu papel no contexto do programa dado. Estas questões testaram o nível que considerámos de Compreensão básica. O terceiro nível continha uma alínea para testar se os alunos conseguiam analisar e compreender um programa completo. Apesar de este nível conter uma questão cognitivamente mais exigente do que as anteriores, consideramos que foram essencialmente testadas habilidades de interpretação e compreensão do código que classificámos ao nível da Compreensão mais avançada. O quarto nível continha uma alínea, para testar se os alunos conseguiam alterar um programa completo. Considerámos que esta questão, poder-se-ia situar ao nível da Aplicação da taxonomia de Bloom. A segunda pergunta, propunha a escrita de um pequeno fragmento de código dentro de um

contexto bem definido (para terminar um programa dado). Na última pergunta, os alunos tinham que implementar um programa completo de raiz.

De notar que, não apenas a situação trimestral como a redução do número de ECTS da disciplina, resultando em menos horas totais do que o habitual, teve também repercussões em termos da avaliação. Nomeadamente, foi retirado o projecto, normalmente realizado extra-aulas e o nível de dificuldade das perguntas, referentes à realização de programas também diminuiu. Por um lado, é nossa convicção que uma organização semestral permite mais tempo para actividades extra lectivas, não existente numa aproximação trimestral. Por outro lado, os objectivos da disciplina também mudaram ligeiramente. Já não era tão importante fazer programas mais elaborados em Assembly, mas antes demonstrar a interconexão do conhecimento oriundo das diversas disciplinas precedentes, tentando contribuir para uma formação de base mais alargada. Consequentemente, o tipo de actividades usadas foi centrado não apenas no desenvolvimento de programas em Assembly mas, a atenção dos alunos foi também dirigida para perguntas teóricas relativas a outro tipo de entendimentos.

6.5.3 Análise de resultados

Relativamente ao factor F2.3 – "As tarefas habitualmente propostas pelos professores apresentam níveis de dificuldades desajustados ao nível cognitivo do aluno", passamos a fazer a análise dos resultados. Um total de 145 alunos (132 do sexo masculino e 13 do sexo feminino) compareceu no exame, porém, apenas 67 (63 do sexo masculino e 4 do sexo feminino) estavam no 1ºano pela primeira vez. Para o propósito do nosso estudo considerámos apenas estes alunos, pois quisemos examinar os alunos que se encontravam em circunstâncias tão homogéneas quanto possível. Para analisar o desempenho dos alunos em cada questão, considerámos toda a resposta com classificação com mais de 50% como satisfatória. Assim, nas quatro alíneas do primeiro grupo de questões (primeiro nível - Conhecimento), a maioria dos alunos obteve resultados satisfatórios.

> *1a) Identifique as linhas onde são declaradas variáveis.*
>
> *1b) Identifique o número total de bytes ocupado por cada variável.*
>
> *1c) Indique o valor dos endereços, em decimal, do primeiro byte ocupado por cada variável.*
>
> *1d) Existe algum ciclo? Em caso afirmativo indique as linhas onde se inicia e qual a condição de saída.*

A percentagem de alunos com resultados satisfatórios nestas questões é apresentada na tabela 6.34.

	Resultados satisfatórios
Questão 1a)	95,3%
Questão 1b)	75,0%
Questão 1c)	68,9%
Questão 1d)	47,3%

Tabela 6.34: Resultados satisfatórios nas questões do nível de Conhecimento

Embora estas questões fossem todas extremamente fáceis, pode-se observar uma diminuição na percentagem de respostas satisfatórias associadas a um aumento ligeiro da dificuldade das perguntas. De salientar o mais baixo resultado obtido nas respostas à questão 1d), provavelmente associado ao facto de esta pergunta envolver o conceito da repetição, geralmente um tópico difícil para muitos alunos.

Nas quatro alíneas do segundo grupo de questões (segundo nível – Compreensão básica), a maioria dos alunos obteve resultados satisfatórios.

> *1e) Qual a diferença entre as instruções mov al,String1[si] e mov al,[si], de forma genérica e no contexto deste programa?*
>
> *1f) Explique qual o objectivo das instruções mov al,String1[si] e mov String2[di],al. Seria possível obter o mesmo efeito através de uma única instrução? Justifique a sua resposta.*
>
> *1g) Qual o objectivo das instruções inc si e add di,1 que surgem nas linhas 16 e 18?*
>
> *1h) Indique uma instrução alternativa a xor si,si.*

A percentagem de alunos com resultados satisfatórios nas perguntas de segundo nível (Compreensão básica) pode ser consultada na tabela 6.35.

	Resultados satisfatórios
Questão 1e)	77,0%
Questão 1f)	75,7%
Questão 1g)	64,9%
Questão 1h)	61,5%

Tabela 6.35: Resultados satisfatórios nas questões do nível de Compreensão (Básica)

Os resultados foram sensivelmente os mesmos que no grupo de questões de primeiro nível. De notar que o nível de dificuldade deste grupo de perguntas também é reduzido, apenas requer um verdadeiro entendimento dos conceitos básicos em questão. Uma

diferença muito mais elevada é, porém, encontrada no terceiro e quarto grupos de questões (terceiro nível – Compreensão avançada) e (quarto nível – Aplicação), como pode ser visto na tabela 6.36.

1i) Explique, de forma sintetizada, o que faz a totalidade de código.

1j) Que alterações teria de realizar se a manipulação fosse efectuada directamente na String1 em vez do resultado ser armazenado na String2?

	Resultados satisfatórios
Questão 1i)	33,1%
Questão 1j)	33,8%

Tabela 6.36: Resultados satisfatórios nas questões do nível de Compreensão (Avançada) e Aplicação

É importante realçar a percentagem mais baixa de resultados satisfatórios nos terceiro e quarto níveis, revelando a dificuldade dos alunos em compreender programas completos ou fazer pequenas alterações, comparativamente à compreensão do papel de instruções simples e/ou isoladas em determinado programa.

Não surpreendentemente, os resultados nos grupos de questões 2 e 3 foram muito mais baixos como ilustrado na tabela 6.37.

	Resultados satisfatórios
Questão 2	15,73%
Questão 3	9,46%

Tabela 6.37: Resultados satisfatórios nas questões do grupo 2 e 3

Estes baixos valores reforçam a grande diferença existente entre a compreensão de uma parte de código, a construção de pequenas partes de código e a construção de um programa completo. Os professores estão familiarizados com alunos que conseguem acompanhar as aulas teóricas de disciplinas de programação, que podem dissecar e compreender programas, mas que são totalmente incapazes de escrever os seus próprios programas. Estes alunos não dominaram todos os processos; conseguem codificar, mas não conseguem produzir um algoritmo (Jenkins, 2002).

Embora esta experiência tenha sido baseada nos resultados do exame final, acreditamos que pode contribuir para uma útil discussão sobre o tipo de actividades usadas em disciplinas de programação, bem como o seu impacto na motivação dos alunos. Como mencionado anteriormente, também as actividades propostas aos alunos durante as aulas foram organizadas de acordo com a taxonomia de Bloom. Esta constituiu uma abordagem completamente diferente das usadas anteriormente, em que a maioria das actividades

consistia em pedir aos alunos que desenvolvessem programas completos para resolverem determinado problema. Como muitos estudantes se revelam incapazes de criar programas, instala-se frequentemente um sentimento de incapacidade, que conduz frequentemente à falta da motivação e abandono. A aproximação usada, incluindo actividades de mais baixo nível de dificuldade, deu a muitos alunos mais fracos um sentimento de confiança pensando que podiam resolver correctamente algumas actividades.

Os impactos positivos que sentimos na aplicação desta abordagem foram medidos, em primeiro lugar, pela taxa de comparência mais elevada, não apenas em sala de aula, mas também em exame, quando comparado com os anos anteriores. Porém, mesmo com resultados interessantes a respeito da assiduidade e participação, o desempenho dos alunos ficou, como habitualmente, muito aquém do desejável. Possivelmente o menor e mais condensado período temporal e a falta de um projecto maior e integrador de vários saberes contribuísse para estes resultados. O número de alunos que compareceu no exame foi mais elevado, não apenas quando comparado com o número de alunos presentes no exame da mesma disciplina em anos anteriores, mas também quando comparado com outras disciplinas que decorreram no mesmo trimestre. Convém salientar que os alunos souberam por antecipação que o exame final teria um grupo de perguntas que não exigiriam a criação de programas completos. Porém, de forma alguma os professores transmitiram a mensagem que responder àquele tipo de perguntas correctamente seria suficiente para obter aprovação (de facto até foi referido o contrário). Foram, no entanto, os alunos que criaram expectativas acerca dessa possibilidade. Porém, no final, os resultados não foram muito diferentes do usual em anos anteriores. Naturalmente que as questões dos grupos 2 e 3 (níveis mais elevados da taxonomia de Bloom) tiveram um peso mais elevado na classificação final e muitos estudantes acabaram por reprovar independentemente de terem boas classificações nas perguntas do grupo 1.

No entanto, o *feedback* recebido por parte dos estudantes, durante as aulas, também indicou um interesse mais elevado demonstrado pelo nível de participação. Muitos estudantes comentaram que estavam mais motivados com esta aproximação. Sentiam-se capazes de compreender algo de programação e de responder correctamente a algumas perguntas, dando-lhes assim algum alento para prosseguirem. Mesmo que o progresso fosse muito lento, os alunos sentiram-se mais confiantes e capazes de aprender alguma coisa de programação. Os alunos comentaram igualmente que ficaram mais cientes que aprender a programar é um processo que consiste em estágios diferentes e não num único estágio impossível de alcançar. Compreenderam que é um processo que exige tempo e muito esforço e métodos intensivos de estudo a fim obter resultados positivos. Outro aspecto reforçado pelos alunos foi respeitante à duplicação de horas de contacto, que permitiu uma maior

proximidade entre aluno e professor, conduzindo a uma maior integração e apoio ao aluno. Igualmente produziu uma maior disponibilidade para o desenvolvimento de habilidades de crítica, autocrítica e discussão.

Consideramos que os alunos devem conseguir desenvolver programas completos e não somente ler e interpretar fragmentos individuais de código. Também acreditamos que os exames e outros métodos da avaliação devem testar esta competência. Porém, pensamos que a aproximação proposta poderia ser usada nas aulas, testes formativos intermédios e em algumas partes dos exames. Nesta disciplina houve a intenção inicial de fazer exames intermédios de acordo com esta abordagem. No entanto, o curto período de tempo disponível para as disciplinas trimestrais inviabilizou este propósito. Pensamos que a estrutura trimestral não é benéfica para diversas disciplinas, em particular naquelas que envolvem a aprendizagem de programação, actividade que requer tempo e maturidade. Pensamos que os alunos necessitam de um período temporal mais alargado para desenvolver as suas habilidades de programação.

Apesar de não ser objectivo nuclear deste estudo, sempre que temos oportunidade aproveitamos para testar os estilos de aprendizagem dos alunos e mais uma vez se comprovaram os perfis identificados em estudos anteriores. Assim, constatou-se que a maioria dos alunos desta amostra era sensorial (80,26%), visual (85,35%), activa (66,39%) e sequencial (71,56%), o que permite, mais uma vez validar o factor F2.1 – A Engenharia Informática/Ciências da Computação atrai alunos com determinado perfil de estilos de aprendizagem. Apesar da disciplina estudada não ser a primeira disciplina de programação ainda tentámos verificar a existência de correlação entre os estilos de aprendizagem dos alunos e os seus resultados à componente prática (programação) desta disciplina. Porém, não foi obtida qualquer correlação entre os perfis de estilos de aprendizagem e os resultados a programação avaliados nesta disciplina.

6.5.4 Conclusões

O principal objectivo deste estudo era o de estudar o factor F2.3 – As tarefas habitualmente propostas pelos professores apresentam níveis de dificuldades desajustados ao nível cognitivo do aluno. Os resultados apresentados levam-nos a confirmar este factor de investigação. Adicionalmente, consideramos que a aproximação seguida neste estudo apresenta claras vantagens pedagógicas, em especial para os alunos mais fracos, ajudando-os a tornar-se mais confiantes e motivados. Para os professores esta abordagem facilita a identificação de dúvidas dos seus alunos relativamente à abordagem tradicional consistindo na escrita de programas completos, desde uma fase inicial, devido às múltiplas dificuldades associadas a esta tarefa.

Em geral, este estudo permitiu-nos recolher as conclusões apresentadas na tabela 6.38:

Estudo D - Conclusões	
Taxomias de Objectivos educacionais	A maioria significativa dos alunos obteve resultados satisfatórios nas questões do nível de Conhecimento e Compreensão básica. Sensivelmente 2/3 dos alunos não obteve resultados satisfatórios nas questões do nível de Compreensão avançada e Aplicação. Apenas 15,73% dos alunos obteve resultados satisfatórios em perguntas que implicavam a escrita de um pequeno fragmento de código dentro de um contexto bem definido. Apenas 9,46% dos alunos obteve resultados satisfatórios numa questão que implicava a escrita de um programa completo.
Estilos de aprendizagem	O perfil mais frequente foi **Sensorial-Visual-Activo-Reflexivo**. Não foi encontrada correlação entre o desempenho a programação e as diferentes categorias dos estilos de aprendizagem.

Tabela 6.38: Conclusões do Estudo D

6.6 Estudo E

O objectivo principal deste estudo foi o de investigar o factor F3, referente aos métodos de estudo utilizados pelos alunos. Desta forma caracterizaram-se os comportamentos de estudo de duas amostras constituídas por alunos matriculados às disciplinas de programação do primeiro ano de licenciaturas em Engenharia Informática, correlacionando posteriormente essas metodologias de estudo com os resultados obtidos nessas disciplinas. Desta forma, apresentam-se dois cenários que servem para verificar a influência dos métodos de estudo no rendimento obtido em disciplinas introdutórias de programação, analisando os diversos níveis de abordagem ao estudo, nomeadamente os enfoques compreensivo, reprodutivo, de envolvimento, de capacidade de organização e as percepções de competência.

6.6.1 Contexto

O estudo ocorreu durante o ano lectivo de 2008/2009 e envolveu alunos matriculados na Licenciatura em Engenharia Informática, da Universidade de Coimbra (LEI-UC) e alunos matriculados na Licenciatura em Engenharia Informática e de Sistemas, do Instituto Superior de Engenharia de Coimbra (LEIS - ISEC). A amostra constituída por alunos da LEI-UC tinha 166 elementos, 155 do sexo masculino e 11 do sexo feminino, com idades compreendidas entre os 18 e 34 anos. A média das idades era de 20,28 anos, mas a mediana situava-se nos 19

anos, a idade comum para alunos do 1º ano. Da totalidade dos 175 alunos da LEIS-ISEC, 168 eram do sexo masculino e 7 do sexo feminino, com idades compreendidas entre os 17 e 48 anos. A média das idades situou-se nos 21,87 anos mas a mediana era de 20 anos. A variabilidade desta amostra é explicada pelo facto da LEIS-ISEC funcionar em dois regimes, diurno e pós-laboral, sendo a maioria dos alunos que frequenta esta último regime, trabalhadores e como tal com idades superiores. Face à diminuta quantidade de elementos do sexo feminino não fizemos diferenciação entre sexos no nosso estudo. De notar que, apesar da amostra inicial conter mais sujeitos, foram retirados desta análise os alunos cuja informação se encontrava incompleta para os objectivos do nosso estudo. As amostras incluíam não apenas caloiros mas também alunos que tinham reprovado anteriormente à 1ª disciplina de programação. Os alunos envolvidos estudaram Python (na LEI-UC) e C (na LEIS-ISEC) na sua primeira disciplina de programação, respectivamente em IPRP (Introdução à Programação e Resolução de Problemas) e AP (Algoritmos e Programação).

6.6.2 Instrumento

O instrumento utilizado neste estudo foi o Inventário de Atitudes e Comportamentos Habituais de Estudo – IACHE (Tavares et al., 2004). Trata-se de um questionário multidimensional dos métodos de estudo constituído por 44 itens. Estes itens estão distribuídos por cinco dimensões. Duas destas dimensões referem-se à forma como os alunos manifestam a sua abordagem à aprendizagem, nomeadamente, profunda (Enfoque compreensivo) ou superficial e memorística (Enfoque reprodutivo). Outra dimensão inclui os itens centrados nas percepções pessoais de capacidade e de realização escolar (Percepções pessoais). As duas últimas dimensões incidem sobre comportamentos motivacionais (Motivação) e de organização (Organização). Desta forma o inventário inclui uma vertente mais comportamental (acções, rotinas diárias, organização do tempo e materiais de estudo), uma vertente mais afectiva-motivacional (compromisso, interesses, envolvimento no estudo e no curso) e uma vertente mais cognitiva (percepções pessoais e atitudes ou enfoques de aprendizagem) (Anexo E). A Tabela 6.39 mostra o conjunto de questões integrantes de cada enfoque.

Enfoques	Questões
Compreensivo	2,8,10,13,16,19,23,26,31,35
Reprodutivo	3,7,9,27,30,33,40,44
Percepções Pessoais	4,15,20,25,28,36,38,43
Motivação	5,12,18,22,29,34,37,42
Organização	1,6,11,14,17,21,24,32,39,41

Tabela 6.39: Questões correspondentes aos Enfoques do questionário IACHE

O inventário inclui também um grupo de oito questões referentes às expectativas académicas e grau de satisfação dos alunos com o curso e a instituição onde funciona. Adicionalmente, existe um outro grupo de questões onde o aluno deve indicar, na sua perspectiva, quais as principais causas subjacentes às possíveis dificuldades de aprendizagem.

O formato da escala utilizado para os alunos demonstrarem o seu grau de concordância com cada pergunta dos diversos enfoques é do tipo *Likert*, de seis pontos, consoante o grau de acordo dos estudantes relativamente à pergunta questionada.

O instrumento de pesquisa foi aplicado em sala de aula. Relativamente aos alunos da LEIS-ISEC, os testes foram aplicados no início da disciplina introdutória de programação AP (início do 1º trimestre) e no final da 3ª disciplina cuja componente prática envolvia programação TI (Tecnologia da Informática) (final do 4º trimestre), em horários previamente combinados entre a investigadora e os professores das disciplinas em causa. Relativamente aos alunos da LEI-UC, os testes foram aplicados no início da 1ª disciplina de programação IPRP (início do 1º semestre) e no final da 2ª disciplina de programação PPP (Princípios de Programação Procedimental) (final do 2º semestre), em horários previamente combinados entre a investigadora e os professores das disciplinas em causa.

6.6.3 Metodologia

Neste estudo, foram utilizadas análises quantitativas descritivas e de correlações, confrontando os resultados provenientes dos dados recolhidos através do questionário IACHE com os resultados das avaliações realizadas nas diversas disciplinas de programação ao longo de 2008/2009. Uma vez que o instrumento utilizado também avalia questões motivacionais e outras referentes às percepções pessoais, estas foram também alvo de análise.

Utilizaram-se também estudos comparativos dentro de cada amostra (LEIS-ISEC e LEI-UC) em diferentes subgrupos para confrontar as mesmas variáveis de forma a verificar se existiam diferenças nesses grupos. Surgiu também a curiosidade de verificar se, dentro de cada grupo, esses comportamentos se mantinham pelo que repetimos os instrumentos de estudo (testes IACHE) realizando um estudo longitudinal, comparando as mesmas variáveis, nos grupos, em dois momentos diferentes.

6.6.4 Análise de Resultados: Amostra LEI-UC

Uma vez que esta escala já foi utilizada em estudos anteriores, encontrando-se aferida para a população com características semelhantes ao nosso público-alvo, não realizámos

nenhum estudo acerca da sua validade interna. Tavares et al. (2004) apresentam os estudos de viabilidade conducentes ao instrumento actual. No entanto fizemos uma análise da sua consistência através do cálculo dos alfas de Cronbach, para cada um dos enfoques, obtendo-se um $\alpha = 0{,}705$. Estes resultados são considerados satisfatórios para a escala total. Analisando as questões integrantes de cada um dos enfoques, os resultados são razoáveis ou bons, nomeadamente $\alpha = 0{,}777$ (para o Enfoque compreensivo), $\alpha = 0{,}844$ (para as Percepções pessoais), $\alpha = 0{,}773$ (para a Motivação) e $\alpha = 0{,}847$ (para a Organização). A excepção vai para o Enfoque reprodutivo $\alpha = 0{,}616$ mais fraco mas, ainda assim, admissível.

6.6.4.1 Análise geral dos enfoques

A Tabela 6.40 ilustra os resultados da estatística descritiva, obtida para cada um dos enfoques, considerando todos os alunos desta amostra.

	N	Mínimo	Máximo	Média	Desvio Padrão
Compreensivo	166	23	55	38,77	6,466
Reprodutivo	166	16	43	29,38	5,125
Percepções Pessoais	166	8	45	25,73	7,883
Motivação	166	11	45	30,78	6,204
Organização	166	11	50	30,47	7,758

Tabela 6.40: Amostra LEI-UC – Estatística descritiva dos Enfoques de todos os alunos

As médias não se distanciam grandemente das verificadas noutros estudos realizados no âmbito das Engenharias, como por exemplo o estudo realizado por Monteiro et al. (2005). No entanto, constata-se que existem grandes variações, verificadas pelos valores mínimos e máximos. De referir que a análise da maioria dos itens é feita pela positiva ou seja uma maior classificação indica maior profundidade de estudo, estudo mais reprodutivo ou memorístico, maior motivação e melhor organização, respectivamente nos enfoques compreensivos, reprodutivos, motivacionais e organizativos. Pelo contrário, no que respeita às percepções pessoais, a análise é feita pela negativa, significando uma classificação maior uma menor percepção de capacidade. Para os enfoques compreensivos e organizativos o valor máximo possível é de 60 pontos, para os níveis reprodutivos, motivacionais e percepções pessoais o máximo é de 48 pontos.

Comparando os resultados do nosso estudo com o estudo de Monteiro et al. (2005) concentrando-nos em especial nos alunos de um curso similar, nomeadamente de Engenharia de Sistemas e Informática, verificamos o seguinte:

- o enfoque compreensivo do nosso estudo apresenta valores médios inferiores (38,77 comparado com 41,12). Atendendo a que o máximo previsto nesta escala é de 60 pontos, podemos afirmar que a média obtida se encontra no limiar de

valores médio-altos, significando que o esforço direccionado para a reflexão e abordagem de estudo profunda dos conteúdos é média-alta.

- o enfoque reprodutivo do nosso estudo apresenta valores médios sensivelmente superiores (29,38 comparado com 28,19). Atendendo a que o máximo previsto nesta escala é de 48 pontos, estes valores significam uma utilização média-elevada de métodos reprodutivos ou de memorização.

- o campo relativo às percepções pessoais dos alunos envolvidos no nosso estudo é mais baixo do que o obtido na outra amostra de comparação (25,73 comparativamente com 29,81). Atendendo a que o nível de percepção pessoal é avaliado pela negativa, significando valores mais elevados piores percepções pessoais, este valor é favorável aos alunos da nossa amostra quando comparado com as outras. Ainda assim, representa uma percepção pessoal negativa mas no limiar do positivo.

- o campo referente à motivação dos alunos envolvidos no nosso estudo apresenta valores médios sensivelmente mais baixos do que a outra amostra de comparação (30,78 comparado com 33,55). Atendendo a que o máximo previsto nesta escala é de 48 pontos, estes valores significam uma motivação na fronteira de valores médio-altos.

- relativamente ao campo da organização os valores encontrados na nossa amostra (30,47) são sensivelmente iguais, em termos médios, aos dos alunos que frequentam o curso similar comparado (30,95). Atendendo a que o máximo previsto nesta escala é de 60 pontos, podemos afirmar que os valores obtidos na nossa amostra se traduzem em níveis de organização médios.

6.6.4.2 Análise dos enfoques entre amostras independentes

As características das amostras estudadas possibilitou ainda a realização de diversas análises, pelo que as particionámos atendendo às diferentes particulares, nomeadamente, tratando os caloiros e os repetentes, bem como os alunos do regime ordinário (R.O.) e do regime trabalhador-estudante (R.T.E.) de forma separada, o que pode ser observado na tabela 6.41, na tabela 6.42, na tabela 6.43, na tabela 6.44 e na tabela 6.45.

	N	Média	Desvio Padrão
Todos (R.O.)	160	38,72	6,374
Todos (R.T.E.)	6	40,17	9,239
Todos (Caloiros)	84	38,93	6,582
Todos (Repetentes)	82	38,61	6,382
Caloiros (R.O.)	84	38,93	6,582
Caloiros (R.T.E.)	Não há Caloiros R.T.E.		
Repetentes (R.O.)	76	38,49	6,172
Repetentes (R.T.E.)	6	40,17	9,239

Tabela 6.41: Amostra LEI-UC – Estatística descritiva do **Enfoque Compreensivo** das várias subamostras

	N	Média	Desvio Padrão
Todos (R.O.)	160	29,38	5,088
Todos (R.T.E.)	6	29,50	6,595
Todos (Caloiros)	84	28,21	5,149
Todos (Repetentes)	82	30,57	4,846
Caloiros (R.O.)	84	28,21	5,149
Caloiros (R.T.E.)	Não há Caloiros R.T.E.		
Repetentes (R.O.)	76	30,66	4,729
Repetentes (R.T.E.)	6	29,50	6,595

Tabela 6.42: Amostra LEI-UC – Estatística descritiva do **Enfoque Reprodutivo** das várias subamostras

	N	Média	Desvio Padrão
Todos (R.O.)	160	25,44	7,662
Todos (R.T.E.)	6	33,50	10,445
Todos (Caloiros)	84	23,94	7,937
Todos (Repetentes)	82	27,57	7,435
Caloiros (R.O.)	84	23,94	7,937
Caloiros (R.T.E.)	Não há Caloiros R.T.E.		
Repetentes (R.O.)	76	27,11	7,029
Repetentes (R.T.E.)	6	33,50	10,445

Tabela 6.43: Amostra LEI-UC – Estatística descritiva do **Enfoque Percepções Pessoais** das várias subamostras

	N	Média	Desvio Padrão
Todos (R.O.)	160	30,86	6,147
Todos (R.T.E.)	6	28,83	7,985
Todos (Caloiros)	84	30,74	6,204
Todos (Repetentes)	82	30,83	6,242
Caloiros (R.O.)	84	30,74	6,204
Caloiros (R.T.E.)	Não há Caloiros R.T.E.		
Repetentes (R.O.)	76	30,99	6,122
Repetentes (R.T.E.)	6	28,83	7,985

Tabela 6.44: Amostra LEI-UC – Estatística descritiva do **Enfoque Motivação** das várias subamostras

	N	Média	Desvio Padrão
Todos (R.O.)	160	30,39	7,666
Todos (R.T.E.)	6	32,50	10,597
Todos (Caloiros)	84	29,80	7,995
Todos (Repetentes)	82	31,16	7,494
Caloiros (R.O.)	84	29,80	7,995
Caloiros (R.T.E.)	Não há Caloiros R.T.E.		
Repetentes (R.O.)	76	31,05	7,281
Repetentes (R.T.E.)	6	32,50	10,597

Tabela 6.45: Amostra LEI-UC – Estatística descritiva do **Enfoque Organização** das várias subamostras

De seguida aplicámos testes t para averiguar os significados obtidos correspondentes às diferenças entre as médias, constatando-se que:

- Podemos assumir como verdadeira a hipótese de que não há diferença entre as médias de todos os alunos do R.O. e todos os alunos do R.T.E., para os enfoques compreensivo, reprodutivo, motivação e organização. Porém, no caso das percepções pessoais, a aplicação do teste de Levene e do teste t para igualdade de médias leva-nos a rejeitar a hipótese de igualdade de médias, pelo que podemos afirmar existir diferenças neste enfoque, obtendo os alunos do R.O. valores mais baixos do que os alunos do R.T.E. Este resultado significa que em geral os alunos do R.T.E. têm piores percepções pessoais do que os alunos do R.O., uma vez que os resultados deste enfoque são analisados pela negativa. Esta informação pode ser consultada na tabela 6.46.

	Teste de Levene para igualdade de variâncias		**Teste t para a igualdade de médias**		
	F	**Sig.**	**t**	**df**	**Sig. (2-*tailed*)**
Compreensivo	0,470	0,494	-0,537	164	0,592
Reprodutivo	1,082	0,300	-0,058	164	0,953
Percepções Pessoais	1,579	0,211	-2,496	164	0,014
Motivação	0,743	0,390	0,783	164	0,435
Organização	1,118	0,292	-0,652	164	0,515

Tabela 6.46: Amostra LEI-UC – Estatística para comparação de médias entre todos os alunos do **R.O.** e todos os alunos do **R.T.E.**

- Podemos assumir como verdadeira a hipótese de que não há diferença entre as médias da totalidade dos caloiros e repetentes, para os enfoques compreensivo, motivação e organização. Porém, no caso do enfoque reprodutivo, mesmo assumindo o teste de igualdade de variâncias, o teste t para igualdade de médias leva a rejeitar a hipótese de igualdade de médias, pelo que podemos afirmar existir diferenças neste enfoque, obtendo os caloiros valores mais baixos do que

os repetentes. Assim, conclui-se que os caloiros utilizam menos técnicas de estudo superficiais ou memorísticas do que os repetentes. No caso das percepções pessoais a aplicação do teste de Levene e teste t leva-nos a rejeitar a hipótese de igualdade de médias. Os resultados obtidos referentes às percepções pessoais são mais elevados nos repetentes do que nos caloiros, significando neste caso piores percepções pessoais dos repetentes relativamente aos caloiros, uma vez que os resultados deste enfoque são analisados pela negativa. Esta informação pode ser consultada na tabela 6.47.

	Teste de Levene para igualdade de variâncias		**Teste t para a igualdade de médias**		
	F	**Sig.**	**t**	**df**	**Sig. (2-*tailed*)**
Compreensivo	0,278	0,599	0,317	164	0,752
Reprodutivo	0,110	0,741	-3,038	164	0,003
Percepções Pessoais	0,541	0,463	-3,044	164	0,003
Motivação	0,017	0,896	-0,94	164	0,925
Organização	0,142	0,706	-1,31	164	0,260

Tabela 6.47: Amostra LEI-UC – Estatística para comparação de médias entre todos os **caloiros** e todos os **repetentes**

- Podemos assumir como verdadeira a hipótese de que não há diferença entre as médias dos alunos caloiros do R. O. e repetentes do R.O., para o enfoque compreensivo, reprodutivo, motivação e organização. Porém, no caso das percepções pessoais, a aplicação do teste de Levene e do teste t leva-nos a rejeitar a hipótese de igualdade de médias, pelo que podemos afirmar existir diferenças neste enfoque. Os caloiros obtiveram valores mais baixos do que os repetentes, o que significa que os caloiros do R. O. apresentam melhores percepções pessoais do que os repetentes do R. O. Esta informação pode ser consultada na tabela 6.48.

	Teste de Levene para igualdade de variâncias		**Teste t para a igualdade de médias**		
	F	**Sig.**	**t**	**df**	**Sig. (2-*tailed*)**
Compreensivo	0,595	0,442	0,437	158	0,663
Reprodutivo	0,295	0,588	-0,3116	158	0,09
Percepções Pessoais	1,465	0,228	-2,659	158	0,002
Motivação	0,125	0,724	-0,255	158	0,799
Organização	0,388	0,534	-1,034	158	0,303

Tabela 6.48: Amostra LEI-UC – Estatística para comparação de médias entre os alunos **caloiros do R.O.** e os alunos **repetentes do R.O.**

- Podemos assumir como verdadeira a hipótese de que não há diferença entre as médias dos alunos repetentes do R.O. e os alunos repetentes do R.T.E., para os

enfoques compreensivo, reprodutivo, motivação e organização. Contudo, no caso das percepções pessoais, a aplicação do teste de Levene e do teste t leva-nos a rejeitar a hipótese de igualdade de médias, pelo que podemos afirmar existirem diferenças neste enfoque, obtendo os alunos repetentes do R.O. valores mais baixos do que os alunos repetentes do R.T.E. Este resultado significa que em geral os alunos repetentes do R.T.E. têm piores percepções pessoais do que os alunos repetentes do R.O., uma vez que os resultados deste enfoque são analisados pela negativa. Esta informação pode ser consultada na tabela 6.49.

	Teste de Levene para igualdade de variâncias		**Teste t para a igualdade de médias**		
	F	**Sig.**	**t**	**df**	**Sig. (2-tailed)**
Compreensivo	0,659	0,419	-0,618	80	0,538
Reprodutivo	1,754	0,189	0,561	80	0,576
Percepções Pessoais	2,645	0,108	-2,069	80	0,042
Motivação	0,800	0,374	0,812	80	0,419
Organização	1,543	0,218	-0,553	80	0,652

Tabela 6.49: Amostra LEI-UC – Estatística para comparação de médias entre os alunos **repetentes do R.O.** e os alunos **repetentes do R.T.E.**

- Não foi possível fazer comparações entre os alunos caloiros do R.O. e os alunos caloiros do R.T.E., nem entre os alunos caloiros do R.T.E. e os alunos repetentes do R.T.E. pois não havia alunos caloiros no R.T.E., eram todos repetentes.

6.6.4.3 Análise dos enfoques entre amostras emparelhadas

Outra análise realizada, para perceber melhor estes dados, consistiu na análise da evolução dos comportamentos dos alunos ao longo do ano lectivo. Para tal, os inquéritos passados no 1º semestre (no início da 1ª disciplina de programação – IPRP) foram novamente passados no final do 2º semestre (no final da 2ª disciplina de programação – PPP). A amostra foi reduzida em virtude de muitos alunos entretanto terem desistido, pelo menos da sua comparência às aulas. De notar que estas amostras têm um número de elementos diferentes das anteriormente apresentadas, pois contêm apenas aqueles alunos que responderam aos dois testes nos dois momentos temporais diferentes, que passamos a designar por inicial e final. Esta informação pode ser consultada na tabela 6.50, na tabela 6.51, na tabela 6.52, na tabela 6.53 e na tabela 6.54.

	N	Média	Desvio Padrão
Todos (Inicial)	74	39,30	6,389
Todos (Final)	74	38,99	6,957
Caloiros (Inicial)	46	40,28	6,517
Caloiros (Final)	46	39,85	7,017
Repetentes (Inicial)	28	37,68	5,932
Repetentes (Final)	28	37,57	6,741
Todos R.O. (Inicial)	74	39,30	6,389
Todos R.O. (Final)	74	38,99	6,957
Caloiros R.O. (Inicial)	46	40,28	6,517
Caloiros R.O. (Final)	46	39,85	7,017
Repetentes R.O. (Inicial)	28	37,68	5,932
Repetentes R.O. (Final)	28	37,57	6,741

Tabela 6.50: Amostra LEI-UC – Estatística descritiva do **Enfoque Compreensivo** das várias subamostras em 2 momentos diferentes (inicial e final)

	N	Média	Desvio Padrão
Todos (Inicial)	74	29,42	5,219
Todos (Final)	74	28,35	5,400
Caloiros (Inicial)	46	28,50	5,124
Caloiros (Final)	46	27,63	5,322
Repetentes (Inicial)	28	30,93	5,106
Repetentes (Final)	28	29,54	5,412
Todos R.O. (Inicial)	74	29,42	5,219
Todos R.O. (Final)	74	28,35	5,400
Caloiros R.O. (Inicial)	46	28,50	5,124
Caloiros R.O. (Final)	46	27,63	5,322
Repetentes R.O. (Inicial)	28	30,93	5,106
Repetentes R.O. (Final)	28	29,54	5,412

Tabela 6.51. Amostra LEI-UC – Estatística descritiva do **Enfoque Reprodutivo** das várias subamostras em 2 momentos diferentes (inicial e final)

	N	Média	Desvio Padrão
Todos (Inicial)	74	24,03	8,119
Todos (Final)	74	24,46	8,123
Caloiros (Inicial)	46	22,46	7,539
Caloiros (Final)	46	23,35	8,141
Repetentes (Inicial)	28	26,61	8,509
Repetentes (Final)	28	26,29	7,897
Todos R.O. (Inicial)	74	24,03	8,119
Todos R.O. (Final)	74	24,46	8,123
Caloiros R.O. (Inicial)	46	22,46	7,539
Caloiros R.O. (Final)	46	23,35	8,141
Repetentes R.O. (Inicial)	28	26,61	8,509
Repetentes R.O. (Final)	28	26,29	7,897

Tabela 6.52: Amostra LEI-UC – Estatística descritiva do **Enfoque Percepções Pessoais** das várias subamostras em 2 momentos diferentes (inicial e final)

	N	Média	Desvio Padrão
Todos (Inicial)	74	31,20	6,493
Todos (Final)	74	30,80	7,264
Caloiros (Inicial)	46	31,65	6,457
Caloiros (Final)	46	31,02	7,565
Repetentes (Inicial)	28	30,46	6,602
Repetentes (Final)	28	30,43	6,861
Todos R.O. (Inicial)	74	31,20	6,493
Todos R.O. (Final)	74	30,80	7,264
Caloiros R.O. (Inicial)	46	31,65	6,457
Caloiros R.O. (Final)	46	31,02	7,565
Repetentes R.O. (Inicial)	28	30,46	6,602
Repetentes R.O. (Final)	28	30,43	6,861

Tabela 6.53: Estatística descritiva do **Enfoque Motivação** das várias subamostras em 2 momentos diferentes (inicial e final)

	N	Média	Desvio Padrão
Todos (Inicial)	74	30,88	7,716
Todos (Final)	74	29,55	8,012
Caloiros (Inicial)	46	30,83	7,301
Caloiros (Final)	46	29,30	7,871
Repetentes (Inicial)	28	30,96	8,492
Repetentes (Final)	28	29,96	8,369
Todos R.O. (Inicial)	74	30,88	7,716
Todos R.O. (Final)	74	29,55	8,012
Caloiros R.O. (Inicial)	46	30,83	7,301
Caloiros R.O. (Final)	46	29,30	7,871
Repetentes R.O. (Inicial)	28	30,96	8,492
Repetentes R.O. (Final)	28	29,96	8,369

Tabela 6.54. Estatística descritiva do **Enfoque Organização** das várias subamostras em 2 momentos diferentes (inicial e final)

De seguida aplicámos testes t para averiguar a diferença entre as médias, constatando-se que:

- Podemos assumir como verdadeira a hipótese de que não há diferença entre as médias de todos os alunos nos dois momentos inicial e final, para os enfoques compreensivo, percepções pessoais e motivação. Porém, no caso da organização, a aplicação dos testes de Levene e teste t leva-nos a rejeitar a hipótese de igualdade de médias, pelo que podemos afirmar que a diferença de 1,324 é significativamente diferente de zero. Esta diferença indica que o nível de organização piorou. Também no caso do enfoque reprodutivo, o teste t tem associado um nível de significância igual a 0,006, o que leva a rejeitar a hipótese de igualdade de médias, pelo que podemos afirmar que a diferença de 1,068 é significativamente diferente de zero. Esta diferença indica que o nível

reprodutivo diminuiu, significando que os alunos passaram a utilizar menos estratégias memorísticas. Esta informação pode ser consultada na tabela 6.55.

	Correlação	Sig.	Dif. Média	t	df	Sig. (*2-tailed*)
Compreensivo	0,814	0,000	0,311	0,651	73	0,517
Reprodutivo	0,814	0,000	1,068	2,834	73	0,006
Percepções Pessoais	0,866	0,000	-0,432	-0,883	73	0,380
Motivação	0,850	0,000	0,405	0,909	73	0,366
Organização	0,896	0,000	1,324	3,161	73	0,002

Tabela 6.55: Amostra LEI-UC – Estatística para comparação de médias iniciais e finais entre todos os alunos

- Podemos assumir como verdadeira a hipótese de que não há diferença entre as médias de todos os caloiros nos dois momentos inicial e final, para os enfoques compreensivo, percepções pessoais e motivação. Porém, no caso do enfoque reprodutivo e da organização, a aplicação do teste de Levene e teste t levam-nos a rejeitar a hipótese de igualdade de médias. Desta forma, podemos afirmar que a diferença de 1,522 das médias do índice organizativo é significativamente diferente de zero, indicando que o nível de organização piorou. Também, a diferença de 0,870 das médias do índice reprodutivo é significativa, indicando que os caloiros passaram a utilizar menos técnicas reprodutivas ou memorísticas. Esta informação pode ser consultada na tabela 6.56.

	Correlação	Sig.	Dif. média	t	df	Sig. (*2-tailed*)
Compreensivo	0,769	0,000	0,435	0,637	45	0,527
Reprodutivo	0,863	0,000	0,870	2,154	45	0,037
Percepções Pessoais	0,877	0,000	-0,891	-1,534	45	0,132
Motivação	0,840	0,000	0,630	1,043	45	0,303
Organização	0,895	0,000	1,522	2,938	45	0,005

Tabela 6.56: Amostra LEI-UC – Estatística para comparação de médias iniciais e finais entre todos os alunos **caloiros**

- Podemos assumir como verdadeira a hipótese de que não há diferença entre as médias de todos os repetentes nos dois momentos inicial e final, para todos os enfoques, nomeadamente compreensivo, reprodutivo, percepções pessoais, motivação e organização. Esta informação pode ser consultada na tabela 6.57.

	Correlação	Sig.	Dif. média	t	df	Sig. (2-*tailed*)
Compreensivo	0,885	0,000	0,107	0,180	27	0,858
Reprodutivo	0,717	0,000	1,393	1,858	27	0,074
Percepções Pessoais	0,846	0,000	0,321	0,370	27	0,714
Motivação	0,874	0,000	0,036	0,056	27	0,956
Organização	0,899	0,000	1,000	1,396	27	0,174

Tabela 6.57: Amostra LEI-UC – Estatística para comparação de médias iniciais e finais entre todos os alunos **repetentes**

- Podemos assumir como verdadeira a hipótese de que não há diferença entre as médias de todos os alunos do R.O. nos dois momentos inicial e final, para os enfoques compreensivo, percepções pessoais e motivação. Contudo, no caso do enfoque reprodutivo e da organização, a aplicação do teste de Levene e do teste t leva-nos a rejeitar a hipótese de igualdade de médias. Desta forma, podemos afirmar que a diferença de 1,324 das médias do índice de organização é estatisticamente diferente de zero, indicando que o nível de organização piorou. Também a diferença de 1,068 das médias do índice reprodutivo desceu, indicando que os caloiros passaram a utilizar menos técnicas reprodutivas ou memorísticas. Esta informação pode ser consultada na tabela 6.58.

	Correlação	Sig.	Dif. média	t	Df	Sig. (2-*tailed*)
Compreensivo	0,814	0,000	0,311	0,651	73	0,517
Reprodutivo	0,814	0,000	1,068	2,834	73	0,006
Percepções Pessoais	0,866	0,000	-0,432	-0,883	73	0,380
Motivação	0,850	0,000	0,405	0,909	73	0,366
Organização	0,896	0,000	1,324	3,161	73	0,002

Tabela 6.58: Amostra LEI-UC – Estatística para comparação de médias iniciais e finais entre todos os alunos do **R.O.**

- Podemos assumir como verdadeira a hipótese de que não há diferença entre as médias dos caloiros do R.O. nos dois momentos inicial e final, para os enfoques compreensivo, percepções pessoais e motivação. No caso do enfoque reprodutivo e da organização, a aplicação do teste de Levene e do teste t leva-nos a rejeitar a hipótese de igualdade de médias. Desta forma, podemos afirmar que a diferença de 1,522 das médias do índice de organização é significativamente diferente de zero, indicando que o nível de organização piorou. Também a diferença de 0,870 das médias do índice reprodutivo desceu, indicando que os caloiros passaram a utilizar menos técnicas reprodutivas ou memorísticas. Esta informação pode ser consultada na tabela 6.59.

	Correlação	Sig.	Dif. média	t	df	Sig. (2-*tailed*)
Compreensivo	0,769	0,000	0,435	0,637	45	0,527
Reprodutivo	0,863	0,000	0,870	2,154	45	0,037
Percepções Pessoais	0,877	0,000	-0,891	-1,534	45	0,132
Motivação	0,840	0,000	0,630	1,043	45	0,303
Organização	0,895	0,000	1,522	2,938	45	0,005

Tabela 6.59: Amostra LEI-UC – Estatística para comparação de médias iniciais e finais entre os alunos **caloiros do R.O.**

- Podemos assumir como verdadeira a hipótese que não há diferença entre as médias dos repetentes do R.O. nos dois momentos inicial e final, para todos os enfoques, nomeadamente compreensivo, reprodutivo, percepções pessoais, motivação e organização. Esta informação pode ser consultada na tabela 6.60.

	Correlação	Sig.	Dif. média	t	df	Sig. (2-*tailed*)
Compreensivo	0,885	0,000	0,107	0,180	27	0,858
Reprodutivo	0,717	0,000	1,393	1,858	27	0,074
Percepções Pessoais	0,846	0,000	0,321	0,370	27	0,714
Motivação	0,874	0,000	0,036	0,056	27	0,956
Organização	0,899	0,000	1,000	1,396	27	0,174

Tabela 6.60: Amostra LEI-UC – Estatística para comparação de médias iniciais e finais entre os alunos **repetentes do R.O.**

- Não foi possível fazer comparações entre os dois momentos com os alunos do R.T.E., nem considerando todos os alunos deste regime, dado o número insuficiente deste tipo de alunos que responderam ao inquérito nos dois momentos. Consequentemente a consideração de apenas os caloiros ou repetentes deste regime foi também comprometida.

Em suma, pode-se constatar que os alunos praticamente não mudaram os seus comportamentos e atitudes de estudo, com excepção dos enfoques organização e reprodutivo. Uma análise mais atenta revela que todos os enfoques desceram ligeiramente, apenas o item percepção pessoal subiu, também ligeiramente, traduzindo uma situação subtilmente mais desfavorável. De notar porém a descida do enfoque reprodutivo com significado positivo.

6.6.4.4 Análise de correlação dos enfoques com os resultados às disciplinas de programação

Outra análise realizada, de interesse particular para o nosso estudo, foi feita correlacionando os resultados referentes às atitudes e comportamentos de estudo com os resultados obtidos às duas primeiras disciplinas de programação realizadas no 1º ano, nomeadamente IPRP e PPP. Os resultados são apresentados na tabela 6.61.

	IPRP	**PPP**
Enfoque Compreensivo	-	-
Enfoque Reprodutivo	-	-
Enfoque Percepções Pessoais	r=-0,507**	r=-0,460**
Enfoque Motivação	r=0,328*	r=0,216*
Enfoque Organização	-	-

*(*ao nível 0,05 (2-tailed), **ao nível 0,01 (2-tailed))*

Tabela 6.61: Amostra LEI-UC – Correlação entre os Enfoques e os resultados a programação

Os resultados obtidos são bastante curiosos, encontrando-se correlações entre os resultados às disciplinas de programação com os itens percepções pessoais e motivação. A correlação negativa obtida pelo cruzamento com o item percepção pessoal, tem neste caso um significado de correlação positiva, na medida em que este enfoque é traduzido pela negativa, ou seja, valores mais elevados representam piores percepções pessoais. Estes resultados significam que, de uma forma bastante significativa (ao nível p=0,01 *2-tailed*), os alunos que têm melhores percepções pessoais são também os que obtêm melhores resultados às primeiras disciplinas de programação. O mesmo se passa em termos motivacionais, embora de uma forma já não tão forte (ao nível p=0,05 *2-tailed*). Assim, os alunos mais motivados são também aqueles que obtêm melhores resultados às primeiras disciplinas de programação. Os resultados obtidos indiciam que os níveis motivacionais e as percepções pessoais são aspectos importantes para obter sucesso a programação.

6.6.4.5 Análise da satisfação dos alunos

Relativamente ao grupo de oito questões referentes às expectativas académicas, o primeiro gráfico (figura 6.12) traduz o grau de satisfação relativamente ao curso (II_1 - disciplinas, II_2 - matérias, II_3 - docentes e II_4 - colegas) e o segundo gráfico (figura 6.13) indica o grau de satisfação relativo à Instituição (II_5 - ambiente geral de trabalho, II_6 - Instituição propriamente dita (campus, espaços, serviços, informação,…), II_7 - participação no estudo/trabalho e II_8 - equipamentos (biblioteca, material, meios informáticos,..)) respectivamente. O formato da escala utilizado para a resposta dos alunos é do tipo *Likert*, de

seis pontos, consoante o grau de concordância (G. C.) dos estudantes relativamente a cada questão.

No que concerne às expectativas/satisfação dos alunos, relativamente ao curso e instituição, podemos dizer que os alunos se encontram, na generalidade, bastante satisfeitos, uma vez que a maioria das respostas obtiveram um grau de concordância (G. C.) entre 3 e 5 pontos. No entanto, podemos afirmar que o único parâmetro que sai menos favorável do padrão de satisfação é referente às expectativas pessoais dos alunos sobre a sua participação no estudo/trabalho (II_7), com a maioria dos alunos a indicar um grau de concordância (G. C.) entre os 3 e 4 pontos. Estes aspectos podem também ser visualizados na figura 6.12, na figura 6.13 e na figura 6.14.

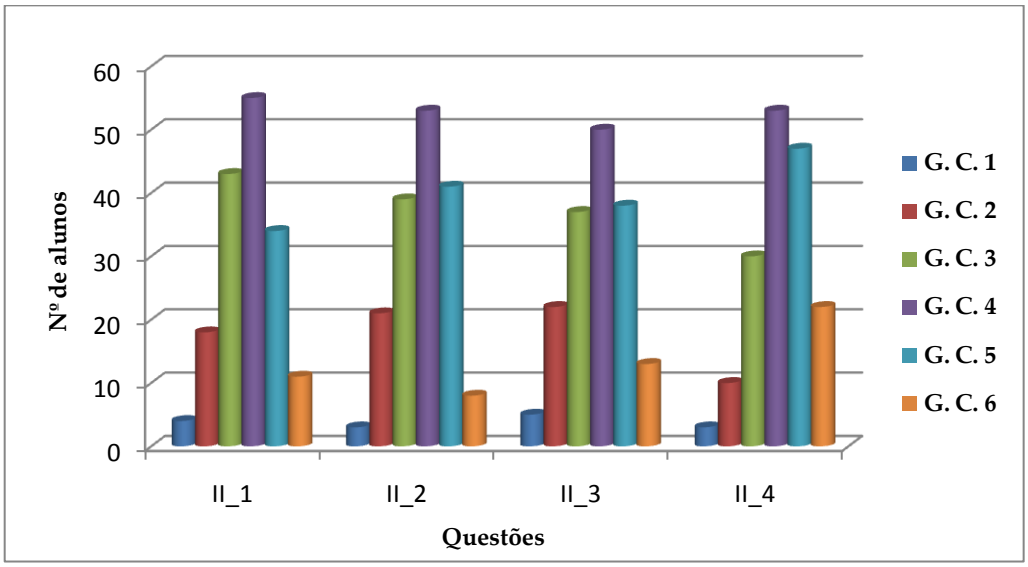


Figura 6.12: Amostra LEI-UC – Grau de satisfação relativamente ao curso

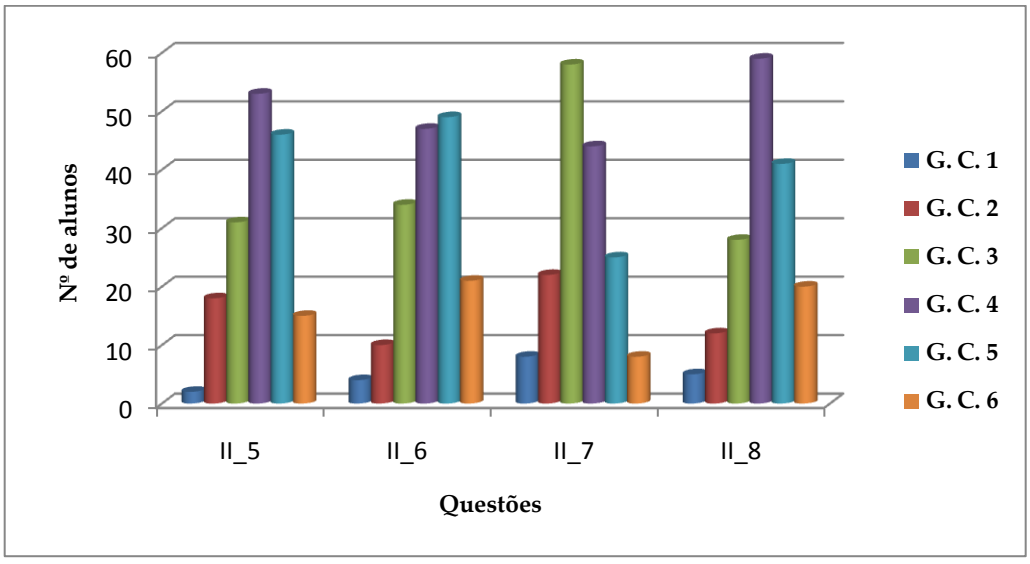


Figura 6.13: Amostra LEI-UC – Grau de satisfação relativamente à Instituição

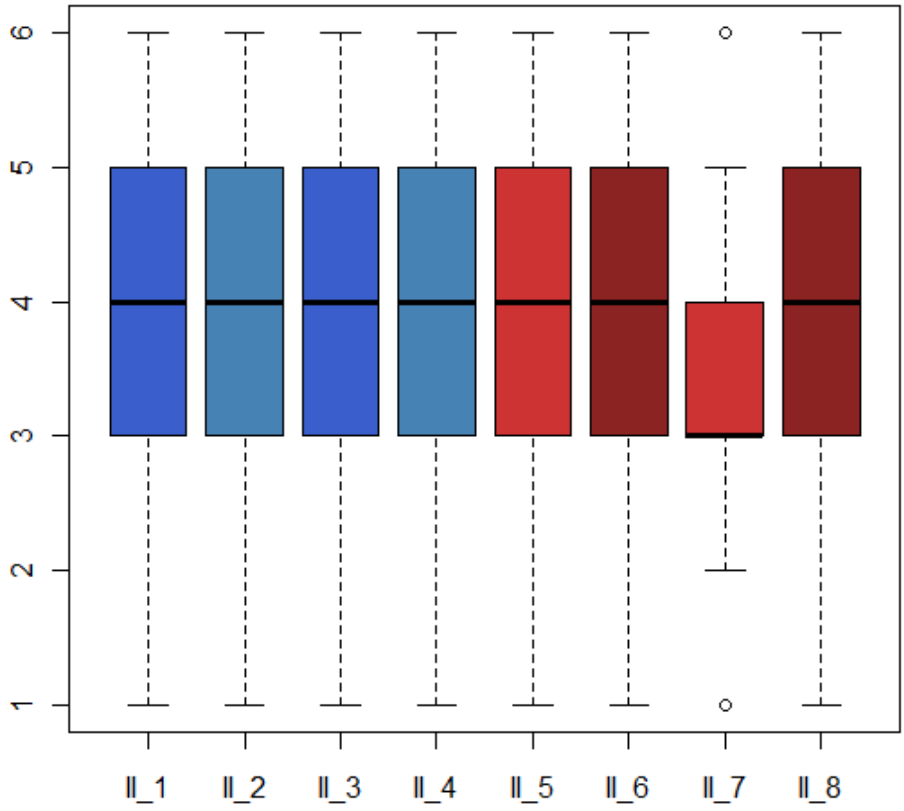


Figura 6.14: Amostra LEI-UC – Grau de satisfação relativamente ao curso e Instituição

Relativamente aos alunos que responderam ao inquérito em dois momentos diferentes, o seu grau de satisfação inicial e final pode ser comparado nos gráficos seguintes (figura 6.15 e figura 6.16). Nesta amostra houve algumas oscilações entre os dois momentos, relativamente ao grau de satisfação com os colegas (II_4) e às expectativas sobre o ambiente geral de trabalho na instituição (II_5). Relativamente ao grau de satisfação com os colegas, embora a maioria dos alunos continue a atribuir uma classificação entre 3 e 5, houve uma redistribuição entre estas classificações, significando que mais alunos diminuíram o seu nível de satisfação relativamente aos colegas. Os alunos desta amostra, quando comparados em dois momentos diferentes revelaram também uma diminuição da satisfação relativa ao ambiente geral de trabalho. Os restantes parâmetros mantiveram-se sensivelmente com os mesmos valores, nos dois momentos.

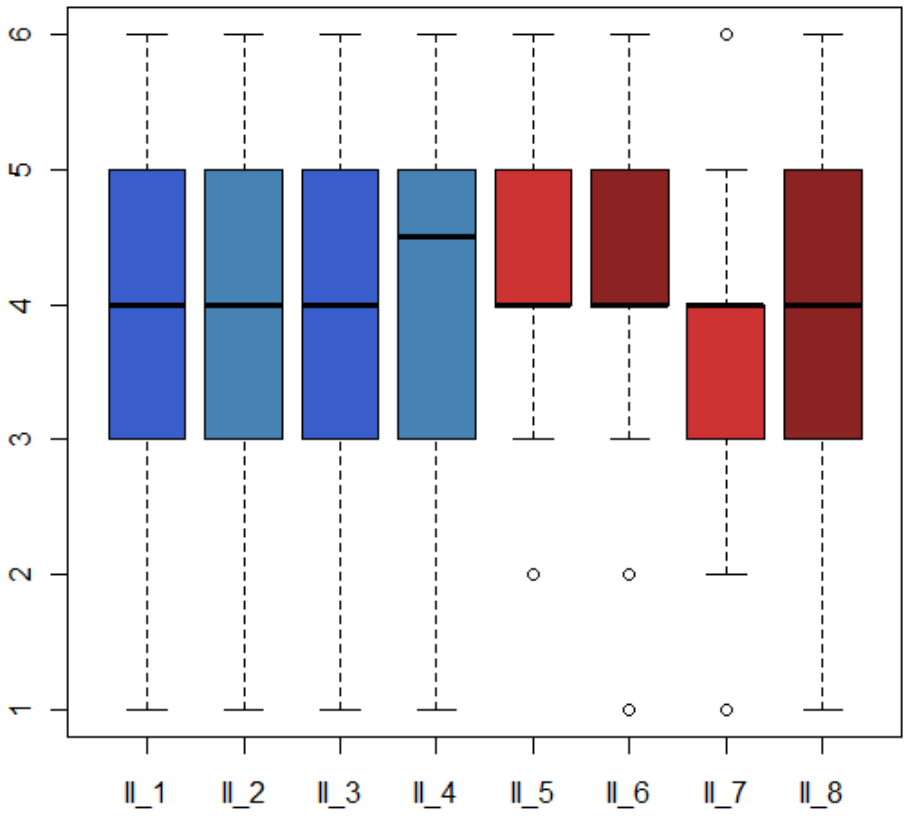


Figura 6.15: Amostra LEI-UC – Grau de satisfação inicial relativamente ao curso e Instituição

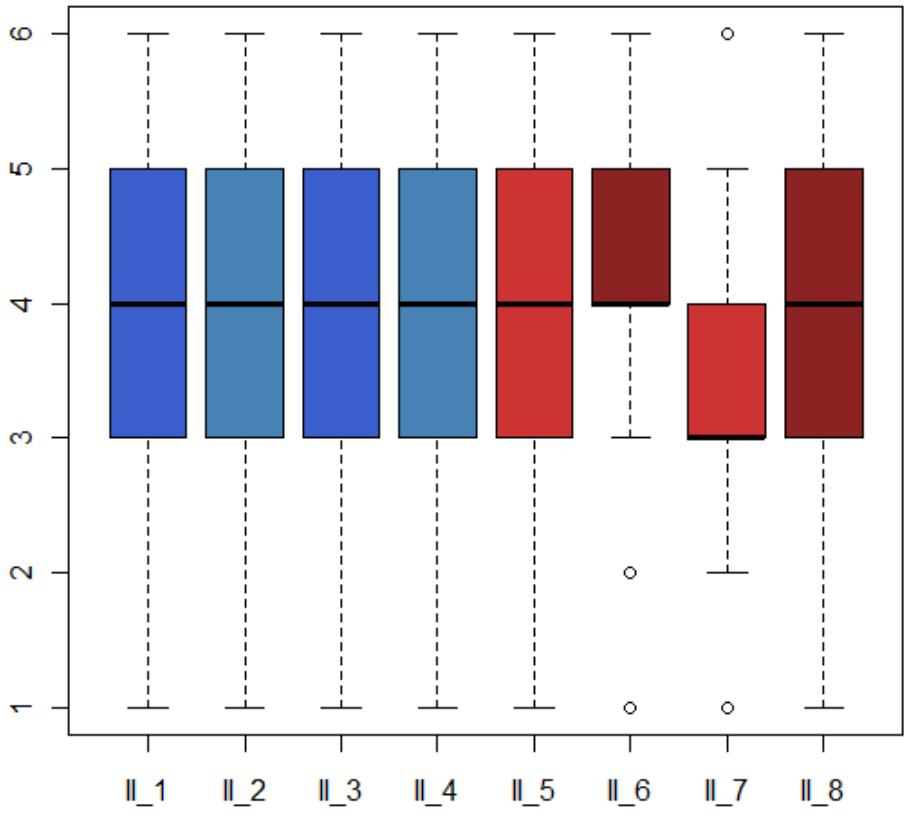


Figura 6.16: Amostra LEI-UC – Grau de satisfação final relativamente ao curso e Instituição

6.6.4.6 Análise das dificuldades dos alunos

O terceiro grupo de questões era relativo às causas das possíveis dificuldades de aprendizagem. Os alunos deveriam escolher uma de entre as seguintes opções: "Falta de bases de conhecimento" (FBC); "Falta de motivação" (FM); "Dificuldades Intelectuais" (DI); "Falta de esforço/persistência pessoal" (FEPP); "Falta de Atenção/Concentração nas Aulas" (FACA); "Competência dos Docentes" (CD); "Dificuldade dos Exames" (DE); "Dificuldades de Adaptação ao Ensino Superior" (DAES); "Métodos de Estudo" (ME); "Falta de Sorte" (FS) ou "Outra razão" (O).

Mesmo sendo dito no teste que os alunos deveriam assinalar a razão mais importante para as possíveis dificuldades de aprendizagem sentidas, uma percentagem considerável (11,80%) indicou diversas causas, que passámos a designar por "D". Dos alunos que indicaram apenas uma razão, as causas mais referidas foram "Falta de bases de conhecimento" (FBC) (31,68%), seguida da "Falta de esforço/persistência pessoal" (FEPP) (19,25%), e "Falta de motivação" (FM) (11,18%). Houve ainda uma percentagem reduzida (3,74%) de alunos que indicou outros factores, mencionando o facto de não se encontrar no curso certo, a falta de tempo, o cansaço, a carga horária excessiva, a concentração do trabalho ou a falta de apoio social. Estes dados podem ser consultados na figura 6.17.

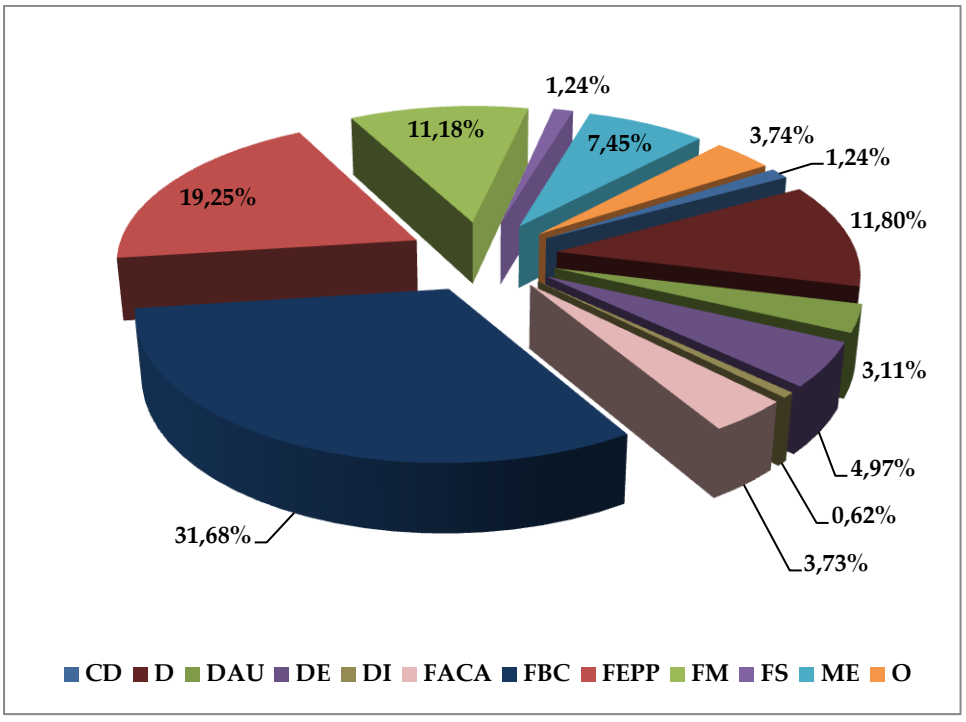


Figura 6.17: Amostra LEI-UC – Tipo de dificuldades

No terceiro grupo de questões relativas às causas das possíveis dificuldades de aprendizagem, os alunos quando solicitados a responderem em dois momentos diferentes, apresentaram causas ligeiramente diferentes. Assim, neste grupo de alunos, (4,11%) indicou inicialmente diversas causas "D". Dos alunos que indicaram apenas uma razão, as causas mais referidas foram a "Falta de bases de conhecimento" (FBC) (36,99%), seguida da "Falta de falta de esforço/persistência pessoal" (FEPP) (23,29%), "Falta de motivação" (FM) (10,96%) e "Falta de atenção e concentração nas aulas" (FACA) (6,85%). Este grupo de alunos indicou, em testes posteriores, de forma mais acentuada (17,57%) diversas causas "D". Dos alunos que indicaram apenas uma razão, as causas mais apontadas foram a "Falta de esforço/persistência pessoal" (FEPP) (22,97%), seguida de, e em percentagens iguais a 9,46%, "Falta de motivação" (FM), "Métodos de estudo" (ME) e "Outros factores" (O). Curiosamente a percentagem de alunos que indicou outros factores subiu de 4,11% (no inquérito inicial) para 9,46% (no inquérito final). Porém, os "Outros factores" mencionados permaneceram os mesmos, nomeadamente o facto de não se encontrarem no curso certo, a falta de tempo, o cansaço, a carga horária excessiva, a concentração de trabalho em termos temporais ou a falta de apoio social. Estes dados podem ser consultados nos gráficos seguintes (figura 6.18 e figura 6.19).

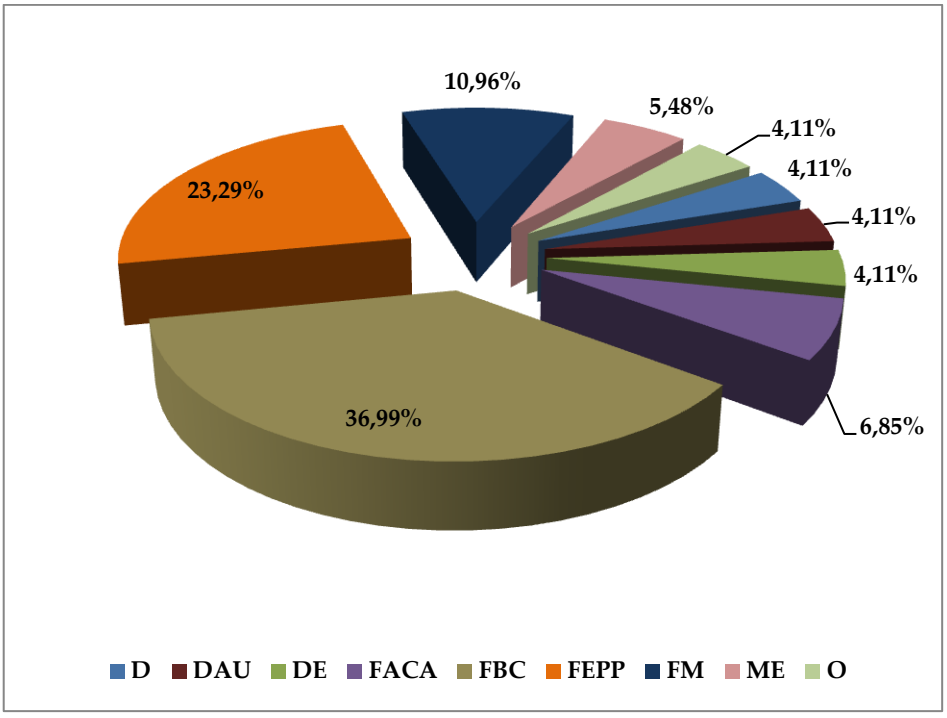


Figura 6.18: Amostra LEI-UC – Tipo de dificuldades iniciais

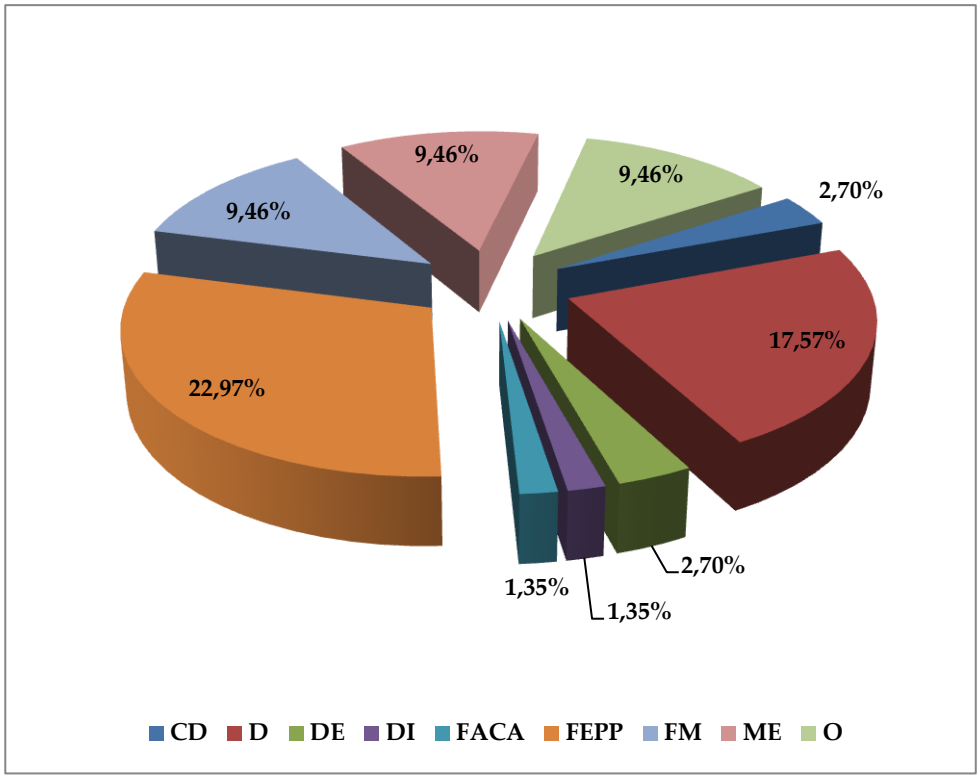


Figura 6.19: Amostra LEI-UC – Tipo de dificuldades finais

6.6.5 Análise de Resultados: Amostra LEIS-ISEC

Mais uma vez procedemos à análise da consistência do inventário IACHE, através do cálculo dos alfas de Cronbach, para cada um dos enfoques, obtendo-se um α=0,785. Estes resultados são considerados satisfatórios para a escala total. Analisando as questões integrantes de cada um dos enfoques, os resultados são razoáveis ou bons, nomeadamente α=0,763 (para o Enfoque compreensivo), α=0,729 (para o Enfoque reprodutivo), α=0,773 (para as Percepções pessoais), α=0,797 (para a Motivação) e α=0,867 (para a Organização).

6.6.5.1 Análise geral dos enfoques

A tabela 6.62 ilustra os resultados obtidos para cada um dos enfoques, considerando todos os alunos desta amostra.

	N	Mínimo	Máximo	Média	Desvio Padrão
Compreensivo	175	20	53	37,57	6,139
Reprodutivo	175	13	41	29,06	5,846
Percepções Pessoais	175	8	44	26,73	6,489
Motivação	175	13	47	29,85	6,430
Organização	175	10	53	28,82	8,159

Tabela 6.62: Amostra LEIS-ISEC – Estatística descritiva dos Enfoques de todos os alunos

Comparando os resultados desta amostra com a outra amostra do nosso estudo (LEI-UC) e com o estudo de Monteiro et al. (2005), concentrando-nos novamente nos alunos de um curso similar, nomeadamente de Engenharia de Sistemas e Informática, verificámos que:

- o enfoque compreensivo desta nossa amostra apresenta valores médios ligeiramente inferiores aos outros (37,57, comparado com 38,77 (LEI-UC) e com 41,12), significando que o esforço direccionado para a reflexão e abordagem mais profunda dos conteúdos é ainda menor nesta amostra, ainda que possa ser considerado médio-alto.

- o enfoque reprodutivo desta nossa amostra apresenta valores médios ligeiramente mais baixos numa situação e ligeiramente mais elevados na outra (29,06, comparado com 29,38 (LEI-UC) e 28,19 do outro estudo). Porém, em termos médios os alunos desta amostra utilizam sensivelmente a mesma quantidade de estratégias memorísticas dos das outras amostras. Ainda assim, atendendo a que o máximo previsto nesta escala é de 48 pontos, estes valores significam também uma utilização média-elevada de métodos reprodutivos ou de memorização.

- o campo relativo às percepções pessoais dos alunos envolvidos nesta amostra é ligeiramente diferente dos obtidos nas outras amostras em comparação (26,73, comparado com 25,73 (LEI-UC) e 29,81 do outro estudo), significando que as percepções pessoais são ligeiramente piores nesta amostra relativamente à nossa outra amostra, e como tal negativas.

- o campo referente à motivação dos alunos desta nossa amostra é inferior a qualquer dos outros estudos (29,85 comparado com 30,78 (LEI-UC) e com 33,55 do outro estudo), significando que os alunos desta amostra são os menos motivados relativamente às outras amostras de comparação. No entanto, uma vez que o máximo previsto nesta escala é de 48 pontos, estes valores ainda poderão ser considerados médios ou elevados.

- relativamente ao campo da organização os valores encontrados nesta amostra são também inferiores a qualquer dos outros estudos (28,82, comparado com 30,47 (LEI-UC) e 30,95 do outro estudo), significando que os alunos desta amostra são os menos organizados relativamente às outras amostras de comparação. No entanto, atendendo a que o máximo previsto nesta escala é de 60 pontos este enfoque representa ainda níveis de organização médios.

6.6.5.2 Análise dos enfoques entre amostras independentes

Também neste cenário particionámos a amostra, tratando os caloiros e os repetentes, bem como os alunos do regime ordinário (R.O.) e os do regime trabalhador-estudante (R.T.E.) de forma separada, o que pode ser observado na tabela 6.63, na tabela 6.64, na tabela 6.65, na tabela 6.66 e na tabela 6.67.

	N	Média	Desvio Padrão
Todos (R.O.)	147	37,31	6,277
Todos (R.T.E.)	28	38,93	5,249
Todos (Caloiros)	85	37,84	6,307
Todos (Repetentes)	90	37,32	6,000
Caloiros (R.O.)	77	37,42	6,344
Caloiros (R.T.E.)	8	41,88	4,454
Repetentes (R.O.)	70	37,20	6,245
Repetentes (R.T.E.)	20	37,75	5,169

Tabela 6.63: Amostra LEIS-ISEC – Estatística descritiva do **Enfoque Compreensivo** das várias subamostras

	N	Média	Desvio Padrão
Todos (R.O.)	147	29,03	6,003
Todos (R.T.E.)	28	29,18	5,041
Todos (Caloiros)	85	29,09	6,185
Todos (Repetentes)	90	29,02	5,542
Caloiros (R.O.)	77	28,86	6,206
Caloiros (R.T.E.)	8	31,38	5,854
Repetentes (R.O.)	70	29,23	5,809
Repetentes (R.T.E.)	20	28,30	4,543

Tabela 6.64: Amostra LEIS-ISEC – Estatística descritiva do **Enfoque Reprodutivo** das várias subamostras

	N	Média	Desvio Padrão
Todos (R.O.)	147	26,36	6,547
Todos (R.T.E.)	28	28,64	5,921
Todos (Caloiros)	85	25,28	6,419
Todos (Repetentes)	90	28,09	6,290
Caloiros (R.O.)	77	25,27	6,423
Caloiros (R.T.E.)	8	25,38	6,823
Repetentes (R.O.)	70	27,56	6,518
Repetentes (R.T.E.)	20	29,95	5,135

Tabela 6.65: Amostra LEIS-ISEC: Estatística descritiva do **Enfoque Percepções Pessoais** das várias subamostras

	N	Média	Desvio Padrão
Todos (R.O.)	147	29,49	6,516
Todos (R.T.E.)	28	31,71	5,708
Todos (Caloiros)	85	30,58	6,534
Todos (Repetentes)	90	29,16	6,289
Caloiros (R.O.)	77	30,12	6,583
Caloiros (R.T.E.)	8	35	4,140
Repetentes (R.O.)	70	28,80	6,417
Repetentes (R.T.E.)	20	30,40	5,798

Tabela 6.66: Amostra LEI-UC – Estatística descritiva do **Enfoque Motivação** das várias subamostras

	N	Média	Desvio Padrão
Todos (R.O.)	147	28,45	8,342
Todos (R.T.E.)	28	30,75	6,937
Todos (Caloiros)	85	30,22	8,788
Todos (Repetentes)	90	27,49	7,321
Caloiros (R.O.)	77	29,60	8,760
Caloiros (R.T.E.)	8	36,25	6,902
Repetentes (R.O.)	70	27,19	7,722
Repetentes (R.T.E.)	20	28,55	5,744

Tabela 6.67: Amostra LEIS-ISEC – Estatística descritiva do **Enfoque Organização** das várias subamostras

Tal como no estudo realizado com a amostra LEI-UC, aplicámos testes t para averiguar a diferença entre as médias nas diversas subamostras, constatando-se que:

- Podemos assumir como verdadeira a hipótese de que não há diferença entre as médias de todos os alunos do R.O. e todos os alunos do R.T.E., para os enfoques compreensivo, reprodutivo, percepções pessoais, motivação e organização. Esta informação pode ser consultada na tabela 6.68.

	Teste de Levene para igualdade de variâncias		**Teste t para a igualdade de médias**		
	F	**Sig.**	**t**	**df**	**Sig. (2-*tailed*)**
Compreensivo	2,733	0,1	-1,279	173	0,203
Reprodutivo	1,548	0,215	-0,120	173	0,905
Percepções Pessoais	0,085	0,770	-1,715	173	0,088
Motivação	1,781	0,184	-1,687	173	0,093
Organização	1,502	0,222	-1,371	173	0,172

Tabela 6.68: Amostra LEIS-ISEC – Estatística para comparação de médias entre todos os alunos do **R.O.** e todos os alunos do **R.T.E**.

- Podemos assumir como verdadeira a hipótese de que não há diferença entre as médias de todos os alunos caloiros e repetentes, para os enfoques compreensivo,

reprodutivo e motivação. Porém, no caso das percepções pessoais e organização, a aplicação do teste de Levene e do teste t leva-nos a rejeitar a hipótese de igualdade de médias, pelo que podemos afirmar existir diferenças nestes enfoques. Os caloiros obtiveram valores mais baixos do que os alunos repetentes relativamente às percepções pessoais. Este resultado significa que em geral os alunos caloiros têm melhores percepções pessoais do que os alunos repetentes. O nível de organização dos caloiros é superior ao dos repetentes. Esta informação pode ser consultada na tabela 6.69.

	Teste de Levene para igualdade de variâncias		**Teste t para a igualdade de médias**		
	F	**Sig.**	**t**	**df**	**Sig. (2-*tailed*)**
Compreensivo	0,078	0,781	0,552	173	0,582
Reprodutivo	0,659	0,418	0,081	173	0,935
Percepções Pessoais	0,084	0,772	-2,921	173	0,004
Motivação	0,001	0,971	1,466	173	0,145
Organização	2,735	0,100	2,241	173	0,026

Tabela 6.69: Amostra LEIS-ISEC – Estatística para comparação de médias entre todos os **caloiros** e todos os **repetentes**

- Podemos assumir como verdadeira a hipótese que não há diferença entre as médias dos alunos caloiros do R.O. e os alunos repetentes do R.O., para os enfoques compreensivo, reprodutivo, motivação e organização. Porém, no caso das percepções pessoais, a aplicação do teste de Levene e do teste t leva a rejeitar a hipótese de igualdade de médias, obtendo os alunos caloiros do R.O. valores mais baixos do que os alunos repetentes do R.O. Este resultado significa que em geral os alunos repetentes do R.O. têm piores percepções pessoais do que os alunos caloiros do R.O. Esta informação pode ser consultada na tabela 6.70.

	Teste de Levene para igualdade de variâncias		**Test t para a igualdade de médias**		
	F	**Sig.**	**t**	**df**	**Sig. (2-*tailed*)**
Compreensivo	0,086	0,770	0,207	145	0,836
Reprodutivo	0,071	0,790	-3,74	145	0,709
Percepções Pessoais	0,003	0,953	-2,139	145	0,034
Motivação	0,045	0,832	1,226	145	0,222
Organização	1,160	0,283	1,763	145	0,080

Tabela 6.70: Amostra LEIS-ISEC – Estatística para comparação de médias entre os alunos **caloiros do R.O.** e os alunos **repetentes do R.O.**

- Podemos assumir como verdadeira a hipótese de que não há diferença entre as médias dos alunos repetentes do R.O. e os alunos repetentes do R.T.E., para os

enfoques compreensivo, reprodutivo, percepções pessoais, motivação e organização. Esta informação pode ser consultada na tabela 6.71.

	Teste de Levene para igualdade de variâncias		**Teste t para a igualdade de médias**		
	F	**Sig.**	**t**	**df**	**Sig. (*2-tailed*)**
Compreensivo	3,538	0,063	-0,360	88	0,720
Reprodutivo	2,344	0,129	0,129	88	0,512
Percepções Pessoais	0,826	0,366	0,366	88	0,134
Motivação	1,111	0,295	0,295	88	0,318
Organização	1,675	0,199	0,199	88	0,465

Tabela 6.71: Amostra LEIS-ISEC – Estatística para comparação de médias entre os alunos **repetentes do R.O.** e os alunos **repetentes do R.T.E.**

- Podemos assumir como verdadeira a hipótese de que não há diferença entre as médias dos alunos caloiros do R.O. e os alunos caloiros do R.T.E., para os enfoques compreensivo, reprodutivo e percepções pessoais. Porém, no caso da motivação e organização, a aplicação do teste de Levene e do teste t leva-nos a rejeitar a hipótese de igualdade de médias, pelo que podemos afirmar existir diferenças nestes enfoques, obtendo os alunos caloiros do R.O. valores mais baixos do que os alunos caloiros do R.T.E. para estes enfoques. Este resultado significa que em geral os alunos caloiros do R.O. têm menos motivação e são menos organizados do que os alunos caloiros do R.T.E. Esta informação pode ser consultada na tabela 6.72.

	Teste de Levene para igualdade de variâncias		**Teste t para a igualdade de médias**		
	F	**Sig.**	**t**	**df**	**Sig. (*2-tailed*)**
Compreensivo	1,438	0,234	-1,934	83	0,057
Reprodutivo	0,227	0,635	-1,097	83	0,276
Percepções Pessoais	0,363	0,548	-0,043	83	0,966
Motivação	1,315	0,255	-2,05	83	0,044
Organização	1,012	0,317	-2,078	83	0,041

Tabela 6.72: Amostra LEIS-ISEC – Estatística para comparação de médias entre os alunos **caloiros do R.O.** e os alunos **caloiros do R.T.E.**

- Podemos assumir como verdadeira a hipótese de que não há diferença entre as médias dos alunos caloiros do R.T.E. e os alunos repetentes do R.T.E., para os enfoques compreensivo, reprodutivo, percepções pessoais e motivação. Porém, no caso da organização, a aplicação do teste de Levene e do teste t leva-nos a rejeitar a hipótese de igualdade de médias, pelo que podemos afirmar existir diferenças neste enfoque, obtendo os alunos caloiros do R.T.E. valores bastante

mais elevados do que os alunos repetentes do R.T.E. Este resultado significa que em geral os alunos caloiros do R.T.E. são mais organizados do que os alunos repetentes do R.T.E. Esta informação pode ser consultada na tabela 6.73.

	Teste de Levene para igualdade de variâncias		**Teste t para a igualdade de médias**		
	F	**Sig.**	**t**	**df**	**Sig. (2-*tailed*)**
Compreensivo	0,019	0,892	1,977	26	0,059
Reprodutivo	0,330	0,571	1,491	26	0,148
Percepções Pessoais	2,140	0,156	-1,939	26	0,063
Motivação	0,336	0,567	2,035	26	0,052
Organização	0,072	0,791	3,028	26	0,005

Tabela 6.73: Amostra LEIS-ISEC – Estatística para comparação de médias entre os alunos **caloiros do R.T.E.** e os alunos **repetentes do R.T.E.**

Esta análise das diferenças permite-nos tecer as seguintes conclusões gerais. Em ambas as amostras (LEI-UC e LEIS-ISEC), considerando a generalidade dos caloiros estes apresentaram melhores percepções pessoais do que os repetentes. Pensamos que este aspecto esteja relacionado com o facto de os repetentes já terem passado por situações de insucesso, afectando as suas percepções pessoais de competência.

Quando considerámos os regimes separadamente já se encontraram algumas variações nas amostras. Nestes casos, numa amostra (LEI-UC) a generalidade dos alunos do R.O. continuou a revelar melhores percepções pessoais do que a generalidade dos alunos do R.T.E. O mesmo se passou com os repetentes do R.O. comparativamente aos repetentes do R.T.E. Não pudemos concluir se este fenómeno também se passava entre caloiros de regimes diferentes devido à inexistência de caloiros no R.T.E. Pensamos que este aspecto possa estar relacionado com o facto de os alunos do R.T.E. terem maior dificuldade em conciliar os seus estudos com o trabalho, fazendo com que os seus projectos académicos sejam frequentemente alvo de insucesso, afectando a sua auto-estima e consequentemente as percepções pessoais de competência. Relativamente à amostra LEIS-ISEC, apenas se notaram diferenças nas percepções pessoais dos caloiros do R.O. relativamente aos repetentes do mesmo regime. Talvez o facto de os alunos já terem sido alvo de insucesso tenha reflexos na sua auto-estima e percepções pessoais de competência, em situações em que os alunos apenas estudam. O facto de este aspecto não ter sido verificado na amostra LEIS-ISEC, entre alunos do R.T.E., pode ser devido ao facto de estes alunos terem um horário especial (pós-laboral) que provavelmente lhes agrada e é conveniente, não afectando o seu desempenho e consequentemente as suas percepções pessoais.

Num caso pontual encontrámos diferenças no que respeita à motivação. Este facto foi encontrado na amostra LEIS-ISEC quando comparámos os caloiros do R.O. com os caloiros

de R.T.E. Enquanto professora de alunos típicos da LEIS-ISEC, a investigadora constata frequentemente a motivação inerente à generalidade dos alunos do R.T.E. É essa motivação que faz com que consigam, na generalidade, assistir às aulas de forma mais assídua e entusiástica do que a maioria dos seus colegas do R.O., mesmo após um dia de trabalho. O facto de se ter verificado esta diferença apenas nos caloiros poderá estar relacionada com o facto de à medida que o tempo vai passando, estes alunos irem reprovando, dada a dificuldade em conciliar o estudo com o trabalho e como tal irem perdendo a motivação.

Também na amostra da LEIS-ISEC, foram encontradas diferenças em termos dos níveis de organização, demonstrando a generalidade dos caloiros ser mais organizados do que a generalidade dos repetentes, os caloiros do R.T.E. serem mais organizados do que os repetentes do R.T.E. e os caloiros do R.O. mais organizados do que os caloiros do R.T.E. Este ultimo aspecto poderia ser justificado com o facto de os alunos do R.O. terem, na generalidade, mais disponibilidade temporal do que os alunos do R.T.E. que na sua maioria acumulam o estudo com o trabalho ficando com menos tempo para proporcionar uma melhor organização. Sendo assim, também os alunos repetentes do R.O. deveriam ser mais organizados do que os alunos repetentes do R.T.E., o que não se verificou. Porém, poderíamos interpretar o facto de ser repetente como estando associado à falta de organização. O que também corroboraria o facto de a generalidade dos caloiros ser mais organizada do que a generalidade dos repetentes, facto que foi verificado. No entanto, temos alguma dificuldade em conseguir explicações mais elucidativas para estes aspectos. Porém, o facto de fazermos afirmações sobre o nível de organização não implica que estejamos a relacioná-lo com o desempenho académico. Estudos conduzidos por outros autores (Monteiro et al., 2005) revelam até que o nível de organização de um indivíduo não está necessariamente relacionado com o seu sucesso académico. Pelo contrário, estes autores concluíram, nesse estudo, que foram os alunos que se mostraram inicialmente menos organizados que foram capazes de realizar mais disciplinas (apesar de neste caso a diferença não ser muito relevante) e com uma melhor média global.

Relativamente ao enfoque reprodutivo, notaram-se diferenças apenas na amostra da LEI-UC, demonstrando a generalidade dos caloiros utilizar menos estratégias reprodutivas do que a generalidade dos repetentes. Talvez o facto de os repetentes utilizarem mais estratégias de estudo superficiais condicione o seu sucesso em disciplinas que exijam uma abordagem ao estudo mais profunda, como as disciplinas de programação. Estas diferenças também foram encontradas entre os caloiros do R.O. e os repetentes do mesmo regime. A impossibilidade de detecção desta diferença entre os caloiros do R.T.E. e os repetentes do mesmo regime pode ser devida à não existência de caloiros no R.T.E que, provavelmente

também se verificaria se houvesse essa possibilidade. Não encontramos explicação para o facto de esta diferença não ter sido detectada na amostra LEIS-ISEC.

6.6.5.3 Análise dos enfoques entre amostras emparelhadas

À semelhança do que fizemos com a amostra anterior, passámos os inquéritos em dois momentos, de forma a poder analisar eventuais alterações dos comportamentos de estudo. No entanto, apesar de muitos alunos responderem aos inquéritos iniciais e finais, o número de alunos que respondeu a ambos foi insuficiente, neste caso, para obter um tratamento estatístico. No entanto, uma análise intuitiva permitiu constatar que os resultados se mantiveram sensivelmente os mesmos.

6.6.5.4 Análise de correlação dos enfoques com os resultados às disciplinas de programação

Outra análise realizada, de interesse particular para o nosso estudo, foi feita correlacionando os resultados referentes às atitudes e comportamentos de estudo com os resultados obtidos às disciplinas de programação realizadas no 1º ano, nomeadamente AP (Algoritmos e Programação), PI (Programação I) e TI (Tecnologia da Informática). Porém, neste caso, não foi possível testar estas correlações, devido ao reduzido número de alunos que realizaram os exames das disciplinas referidas e que simultaneamente tinham respondido ao inquérito. Pensamos que este resultado possa ser justificado pela organização curricular em trimestres ocorrida nesta instituição neste ano lectivo. Constatámos que, trimestre após trimestre o número de alunos nos exames sofreu um decréscimo significativo, veja-se esta análise em Gomes e Mendes (2009).

6.6.5.5 Análise da satisfação dos alunos

Relativamente ao grupo de oito questões referentes às expectativas académicas, o primeiro gráfico (figura 6.20) traduz o grau de satisfação relativamente ao curso (II_1 - disciplinas, II_2 - matérias, II_3 - docentes e II_4 - colegas) e o segundo gráfico (figura 6.21) indica o grau de satisfação relativo à Instituição (II_5 - ambiente geral de trabalho, II_6 - Instituição propriamente dita (campus, espaços, serviços, informação,…), II_7 - participação no estudo/trabalho e II_8 - equipamentos (biblioteca, material, meios informáticos,..)) respectivamente. O formato da escala utilizado para a resposta dos alunos é do tipo *Likert*, de seis pontos, consoante o grau de concordância (G. C.) dos estudantes relativamente a cada questão.

No que respeita às expectativas/satisfação dos alunos, relativamente ao curso e instituição, podemos dizer que os alunos se encontram, na generalidade, bastante satisfeitos, uma vez que a maioria das respostas obtiveram um grau de concordância (G. C.) entre 3 e 5 pontos ou entre 4 e 5 pontos. No entanto podemos afirmar que os parâmetros correspondentes às disciplinas (II_1) e matérias (II_2) do curso e às expectativas pessoais dos alunos sobre a sua participação no estudo/trabalho (II_7) saem menos favoráveis, com a maioria a indicar um grau de concordância (G. C.) entre os 3 e 4 pontos. Estes aspectos podem também ser visualizados na figura 6.20, na figura 6.21 e na figura 6.22.

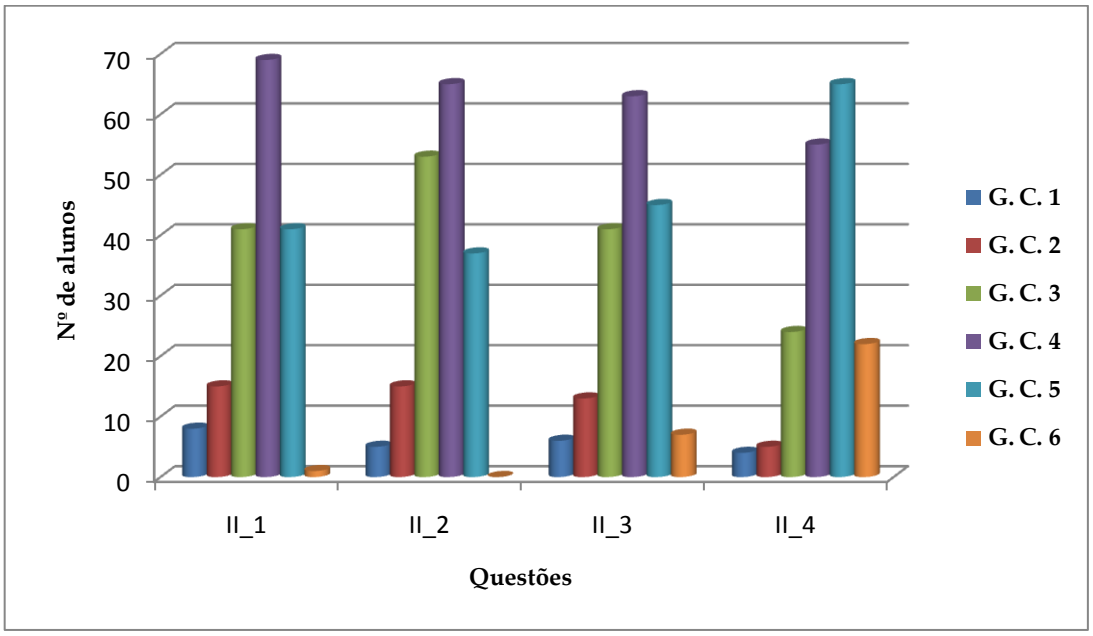


Figura 6.20: Amostra LEIS-ISEC – Grau de satisfação relativamente ao curso

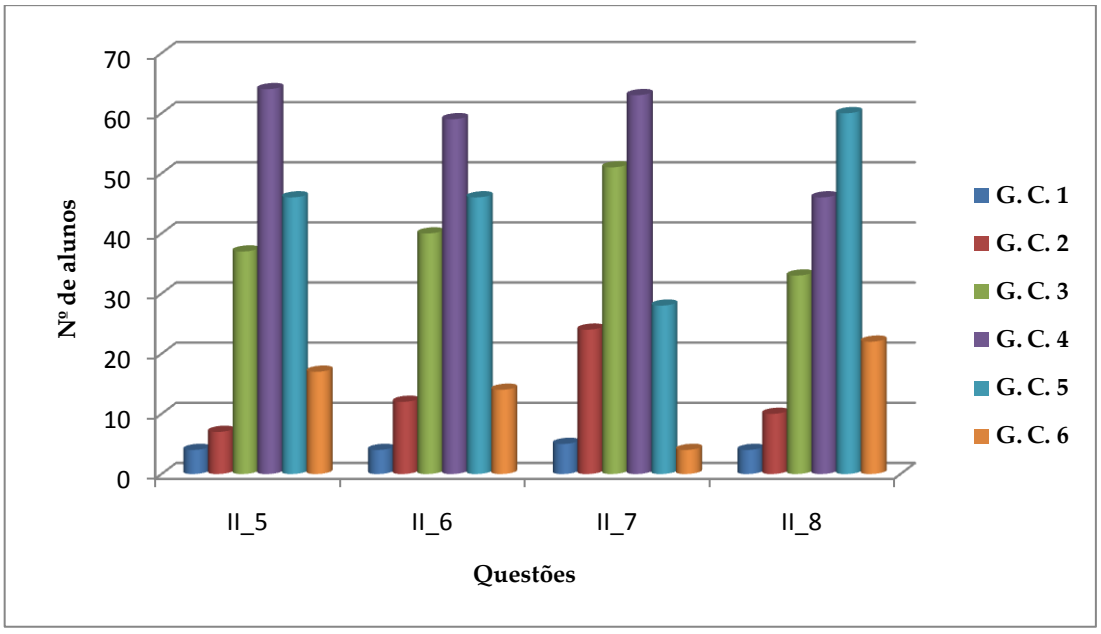


Figura 6.21: Amostra LEIS-ISEC – Grau de satisfação relativamente à Instituição

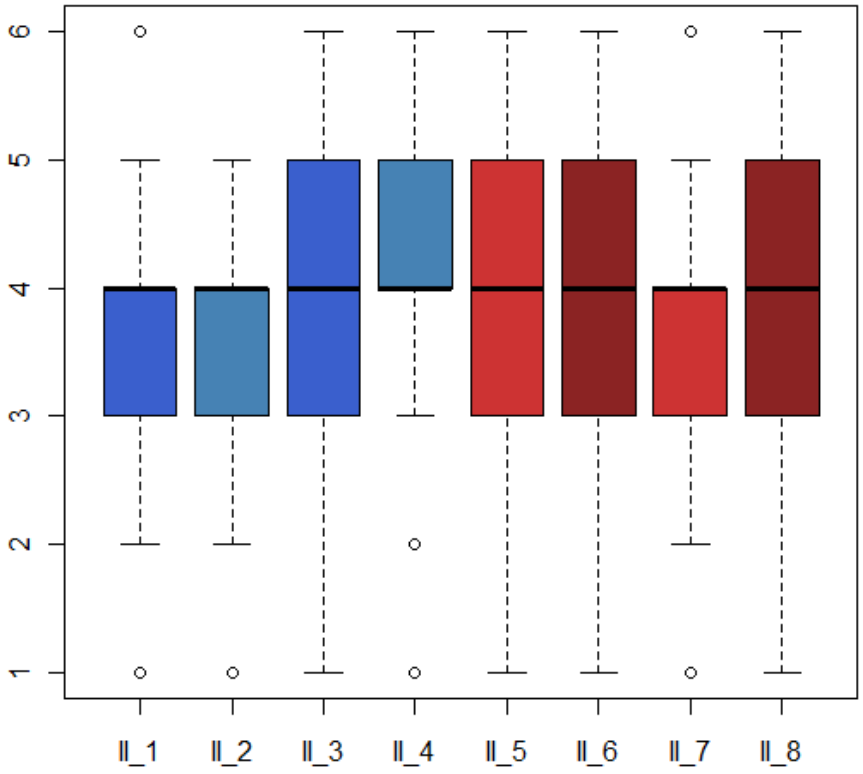


Figura 6.22: Amostra LEIS-ISEC – Grau de satisfação relativamente ao curso e Instituição

Mais uma vez não foi possível, com esta amostra, fazer uma análise comparativa longitudinal referente à evolução dos níveis de satisfação, devido à insuficiência de alunos comuns que realizaram os inquéritos nos dois momentos temporais.

Resumindo, podemos tecer as seguintes conclusões gerais. Relativamente às expectativas académicas, pensamos que os estudos revelam que, em qualquer das amostras os resultados são médios ou elevados, nas duas amostras LEI-UC e LEIS-ISEC. O aspecto que conseguiu pontuações mais baixas, nas duas amostras foi o referente à satisfação do aluno com a sua participação no estudo/trabalho.

6.6.5.6 Análise das dificuldades dos alunos

Analisámos também as dificuldades expressas pelos alunos através do terceiro grupo de questões. Embora se tenha pedido aos alunos para indicarem a razão mais importante das possíveis causas de aprendizagem, uma percentagem significativa (40,46%) indicou diversas causas "D". Dos alunos que indicaram apenas uma razão, as causas mais referidas foram "Falta de esforço/persistência pessoal" (FEPP) (12,72%), "Métodos de estudo" (ME) (9,25%) e "Falta de motivação" (FM) (8,67%). Estes dados podem ser consultados na figura 6.23. Houve ainda uma percentagem reduzida (4,62%) de alunos que indicou outros factores,

mencionando em especial a falta de tempo. Este aspecto foi apenas referido pelos alunos do regime trabalhador-estudante.

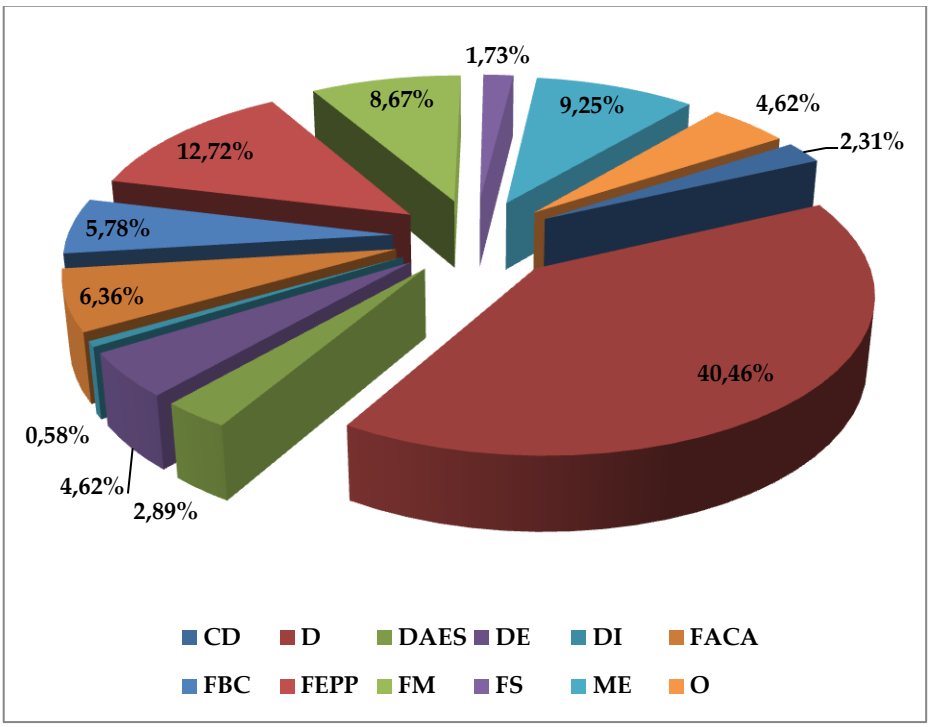


Figura 6.23: Amostra LEIS-ISEC – Tipo de dificuldades

No terceiro grupo de questões relativas às causas das possíveis dificuldades de aprendizagem, os alunos quando solicitados a responder em dois momentos diferentes, apresentaram causas ligeiramente diferentes. Assim, neste grupo de alunos, (35%) indicou inicialmente diversas causas "D". Dos alunos que indicaram apenas uma razão, as causas mais referidas foram a "Competência dos docentes" (CD) (15%), seguida em igual percentagem de "Falta de esforço/persistência pessoal" (FEPP) (10%), "Falta de atenção e concentração nas aulas" (FACA) (10%),"Falta de sorte" (FS) (10%) e "Outros motivos" (O) (10%). Este grupo de alunos indicou, em testes posteriores, de forma significativamente mais acentuada (70%) diversas causas "D". Dos alunos que indicaram apenas uma razão, as causas mais apontadas foram os "Métodos de estudo" (ME) (10%), seguido em igual percentagem de "Falta de esforço/persistência pessoal" (FEPP) (5%), "Falta de atenção e concentração nas aulas" (FACA) (5%), "Dificuldade dos exames" (DE) (5%) e "Outros motivos" (O) (5%). De notar o aumento significativo de alunos que indicou diversos factores e, para além de os alunos continuarem a mencionar a "Falta de esforço/persistência pessoal" e a "Falta de atenção e concentração nas aulas", deixaram de mencionar a "Competência dos docentes" e a "Falta de sorte" como factores exclusivos dos seus insucessos. Porém, mencionaram um novo factor não referido inicialmente, a dificuldade dos exames. Relativamente aos "outros motivos" os alunos mencionaram essencialmente os mesmos

motivos nos dois momentos temporais, nomeadamente a falta de tempo, em especial os alunos do regime trabalhador-estudante, acrescido, nalguns casos, pelo facto de serem pais. Estes dados podem ser consultados nos gráficos seguintes (figura 6.24 e figura 6.25).

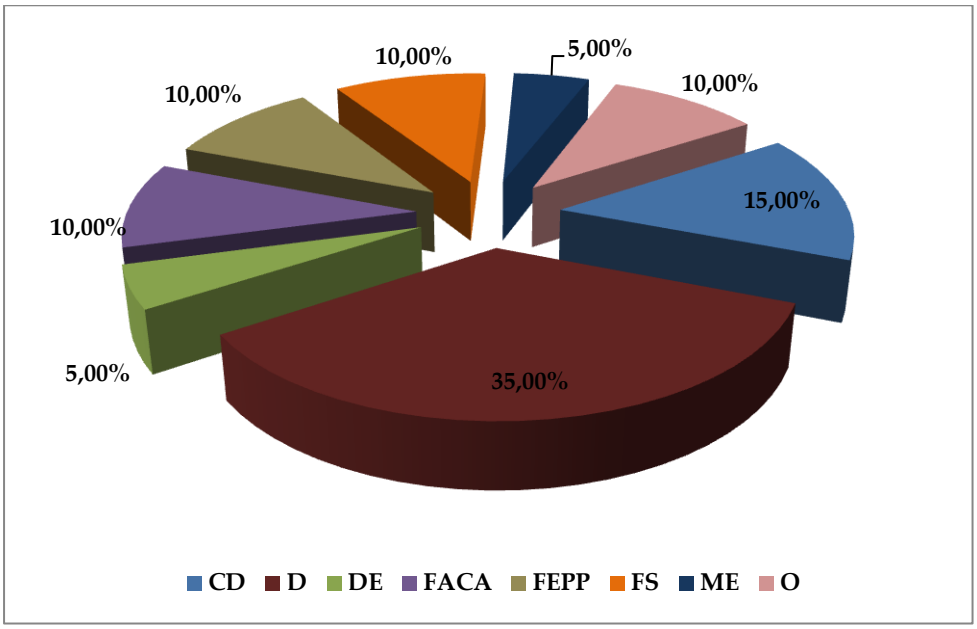


Figura 6.24: Amostra LEIS-ISEC – Tipo de dificuldades iniciais

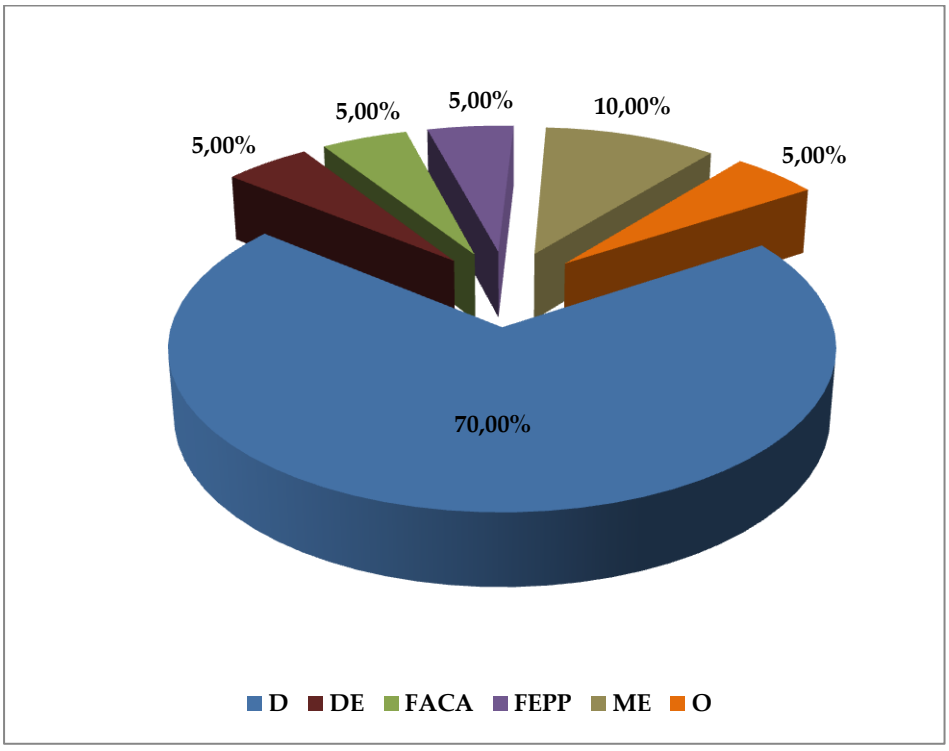


Figura 6.25: Amostra LEIS-ISEC – Tipo de dificuldades finais

Em suma, podemos concluir que os alunos apontam várias razões para as suas dificuldades de aprendizagem, porém, a maioria indicada por ambas as amostras LEI-UC e LEIS-ISEC, vão para a "Falta de esforço/persistência pessoal" e a "Falta de motivação", embora a "Falta de bases de conhecimento e os "Métodos de estudo" também sejam reportados por uma razoável percentagem de alunos. Este é mais um elemento que coloca a tónica na importância das questões motivacionais e de percepção pessoal de auto-eficácia, persistência e auto-estima.

6.6.6 Conclusões

O principal objectivo deste estudo era o de estudar o factor F3 – Os métodos de estudo utilizados pelos alunos não são os mais adequados para a aprendizagem em geral e da programação em particular. Apesar de o instrumento utilizado revelar classificações elevadas na generalidade dos parâmetros por ele avaliados, nomeadamente no que respeita à abordagem à aprendizagem, às percepções pessoais de capacidade e de realização escolar, aos comportamentos motivacionais e de organização, não temos parâmetros para poder aceitar ou rejeitar esta hipótese. De notar também que o facto de alguns desses aspectos se traduzirem por classificações média-altas, não têm necessariamente significados positivos, nomeadamente no que respeita aos aspectos de percepções pessoais e enfoque reprodutivo. Adicionalmente, não temos conhecimento de nenhum elemento que nos permita afirmar se determinada classificação obtida em determinado parâmetro revela um tipo de abordagem ao estudo e comportamento adequado a determinado assunto ou disciplina. O que podemos afirmar é que os resultados obtidos não se distanciam grandemente de outros obtidos com públicos-alvo com características similares ao nosso. Porém, face ao insucesso generalizado em termos de resultados obtidos a programação pela generalidade desse tipo de público, talvez possamos afirmar que embora esses parâmetros sejam bons ou aceitáveis para a generalidade dos assuntos não serão suficientes para obter sucesso a disciplinas de programação.

Este estudo levou-nos ainda a acrescentar o factor F4 – As percepções pessoais dos alunos são em geral muito baixas, insuficientes para enfrentar as exigências das disciplinas de programação. Os resultados apresentados levam-nos a considerar que este factor de investigação possa ser verdadeiro, uma vez que se verificou na amostra estudada. Isto porque, na amostra em que tínhamos elementos suficientes para obter resultados estatísticos foram encontradas correlações entre os níveis motivacionais e o desempenho às disciplinas de programação do 1º ano, e obtidas correlações ainda mais fortes entre estas últimas e as percepções pessoais de competência dos alunos.

Em geral, este estudo permitiu-nos recolher as conclusões, sobre as metodologias, atitudes e comportamentos de estudo apresentadas na tabela 6.74.

Estudo E - Conclusões	
Enfoque Compreensivo	Médio-alto. Não foi encontrada correlação entre o desempenho a programação e o enfoque compreensivo, em ambas as amostras.
Enfoque Reprodutivo	Médio-alto. Em geral, os caloiros revelaram menos abordagens memorísticas do que os repetentes, na amostra LEI-UC. Não foi encontrada correlação entre o desempenho a programação e o enfoque reprodutivo, em ambas as amostras.
Enfoque Percepções Pessoais	Médio-Baixo. Em geral, os caloiros revelaram melhores percepções pessoais do que os repetentes, em ambas as amostras. Em geral, os alunos do R.O. revelaram melhores percepções pessoais do que os alunos do R.T.E., na amostra LEI-UC. Os caloiros do R.O. revelaram melhores percepções pessoais do que os repetentes do R.O., na amostra LEIS-ISEC. Na amostra LEI-UC, foi encontrada correlação forte entre o desempenho na disciplina introdutória de programação (IPRP) e as percepções pessoais dos alunos da amostra. Na amostra LEI-UC, foi encontrada correlação forte entre o desempenho na segunda disciplina de programação (PPP) e as percepções pessoais dos alunos da amostra.
Enfoque Motivação	Médio-alto. Os caloiros do R.T.E. revelaram melhores níveis motivacionais do que os caloiros do R.O., na amostra LEIS-ISEC. Na amostra LEI-UC, foi encontrada correlação entre o desempenho na disciplina introdutória de programação (IPRP) e os níveis motivacionais dos alunos da amostra. Na amostra LEI-UC, foi encontrada correlação entre o desempenho na segunda disciplina de programação (PPP) e os níveis motivacionais dos alunos da amostra.
Enfoque Organização	Médio. Em geral, os caloiros mostraram melhores níveis de organização do que os repetentes, na amostra LEIS-ISEC. Não foi encontrada correlação entre o desempenho a programação e o nível de organização, em ambas as amostras.

(cont.)

Estudo E – Conclusões (Continuação)	
Expectativas académicas	Médias-altas. A expectativa que apresenta valores inferiores diz respeito, nas duas amostras, à satisfação do aluno com a sua participação no estudo/trabalho.
Principais dificuldades de aprendizagem	São apontadas várias razões, em ambas as amostras. Em geral, a maioria dos alunos, em ambas as amostras LEI-UC e LEIS-ISEC, refere a "Falta de esforço/persistência pessoal" e a "Falta de motivação". A "Falta de bases de conhecimento" e os "Métodos de estudo" também são reportados por uma razoável percentagem de alunos.

Tabela 6.74: Conclusões do estudo E

6.7 Estudo F

Este estudo teve como principal objectivo avaliar o impacto de uma nova metodologia de ensino, nos resultados finais obtidos a programação. Esta nova metodologia consistia na realização de actividades de programação em C, no início de algumas aulas práticas, relativas às matérias leccionadas nas aulas anteriores. Para tal foi escolhida a disciplina de PPP (Princípios de Programação Procedimental) onde os alunos estudaram a linguagem de programação C. Esta disciplina foi antecedida por outra disciplina de programação, IPRP (Introdução à Programação e Resolução de Problemas) onde os alunos estudaram Python. Desta forma as aulas iniciais de PPP consistiram essencialmente na tradução das estruturas de controlo básicas e regras de sintaxe de Python para C. Assim, as actividades referidas foram realizadas a partir do momento em que os alunos tinham o conhecimento suficiente da linguagem C para realizar tais actividades. A maioria das actividades consistia na análise de programas, após o que os alunos tinham de responder a questões sobre as codificações fornecidas, indicando a veracidade de determinadas afirmações ou escolhendo a afirmação verdadeira, referente à codificação analisada, de entre um conjunto de afirmações. Outras actividades consistiam em o aluno escolher qual, de entre várias codificações é que satisfazia determinados requisitos ou correspondia a determinada saída. Outras actividades consistiam em os alunos completarem pequenos pedaços de código, escolhendo o código em falta de entre um conjunto de opções fornecidas. O impacto referente à realização destas actividades seria avaliado através da correlação entre a quantidade e qualidade das actividades realizadas com os desempenhos obtidos a uma disciplina de programação. Consequentemente, experimentou-se também se o facto de os alunos terem uma abordagem de ensino diferente, com actividades organizadas, tinha influência nos resultados de aprendizagem nessa disciplina.

Aproveitando ainda a informação existente sobre as metodologias e comportamentos de estudo desta amostra (obtida no Estudo E), tentou-se perceber a relação entre estes aspectos e o envolvimento dos alunos em determinadas actividades.

6.7.1 Contexto

O estudo ocorreu durante o 2º semestre do ano lectivo de 2008/2009 e envolveu alunos matriculados na Licenciatura em Engenharia Informática, da Universidade de Coimbra (LEI-UC). A amostra incluiu 130 alunos, sendo 116 do sexo masculino e os restantes do sexo feminino, pelo que não fizemos diferenciação por sexo. De notar que, apesar da amostra inicial conter mais sujeitos, foram retirados desta análise os alunos cuja informação se encontrava incompleta. Os estudantes envolvidos incluíam não apenas caloiros, mas também alunos que já tinham reprovado à 1ª disciplina de programação.

6.7.2 Instrumento

Os instrumentos utilizados neste estudo foram o Inventário de Atitudes e Comportamentos Habituais de Estudo – IACHE (Anexo E), já descrito neste documento e diversas fichas de actividades, desenvolvidas pela investigadora, a realizar no início de cada aula laboratorial ao longo de todo o semestre (Anexo F.1 - Anexo F.9).

6.7.3 Metodologia

A metodologia principal utilizada neste estudo foi a experiência, de forma a permitir esclarecer a relação causal entre determinado método de ensino e os resultados a programação. Para tal constituíram-se dois grupos – um grupo experimental que seria submetido à realização de actividades no início de cada aula prática e um grupo de controlo onde essas actividades não seriam realizadas.

Para a realização das actividades, foram seleccionadas várias turmas (cujos alunos integravam o grupo experimental), onde, semanalmente no início de cada aula laboratorial eram realizadas actividades de avaliação formativa, referentes aos conteúdos praticados em aulas anteriores. As actividades eram disponibilizadas e realizadas através da actividade Teste da plataforma *Moodle*, nos horários das aulas práticas das turmas seleccionadas.

De forma a ter o máximo de controlo sobre a variação da variável independente e controlar a influência de outras variáveis, realizámos também inquéritos sobre as metodologias de estudo. Para tal, o questionário IACHE foi aplicado aos alunos no início do ano lectivo, na 1ª disciplina de programação (IPRP). Posteriormente, esse instrumento foi novamente aplicado na 2ª disciplina de programação (PPP), de forma a verificar se houve

mudanças de comportamentos relativamente aos métodos de estudo adoptados pelos alunos inicialmente. Os resultados provenientes dos dados recolhidos através deste questionário foram confrontados com os resultados das avaliações realizadas nas duas disciplinas de programação (IPRP e PPP) bem como com os resultados das actividades realizadas. Em qualquer das situações utilizámos análises quantitativas descritivas e de correlações.

6.7.4 Análise de Resultados

Em primeiro lugar procedeu-se à caracterização da amostra em termos de cada um dos enfoques. A presente amostra é uma subamostra da descrita no Estudo E (amostra LEI-UC) anteriormente apresentado na secção 6.6.4. Embora a amostra tenha sido reduzida, pois o número de alunos que realizaram as actividades foi um subconjunto da outra amostra, os resultados referentes a esta subamostra seguem sensivelmente os mesmos padrões da amostra maior. A tabela 6.75 ilustra os resultados da estatística descritiva, obtida para cada um dos enfoques, considerando os alunos desta amostra.

	N	Mínimo	Máximo	Média	Desvio Padrão
Compreensivo	130	23	55	39,18	6,196
Reprodutivo	130	18	43	29,45	4,988
Percepções Pessoais	130	8	45	25,15	7,573
Motivação	130	16	45	31,19	6,247
Organização	130	11	50	30,56	8,327

Tabela 6.75: Estatística descritiva dos Enfoques

A tabela 6.76, a figura 6.26 e a figura 6.27 apresentam a estatística descritiva e respectiva ilustração referentes às actividades realizadas.

	N	Mínimo	Máximo	Média	Desvio Padrão
Actividade 1	55	7,00	20,00	14,69	3,76
Actividade 2	8	0,83	10,00	6,04	3,53
Actividade 3	42	1,66	15,71	7,60	3,66
Actividade 4	20	1,25	15,00	12,31	4,26
Actividade 5	18	2,33	17,67	10,98	5,01
Actividade 6	13	0,67	18,00	7,21	5,16
Actividade 7	11	0,80	20,00	9,31	6,26
Actividade 8	2	8,04	9,29	8,67	0,88
Actividade 9	3	12,00	20,00	14,83	4,48

Tabela 6.76: Estatística descritiva das actividades

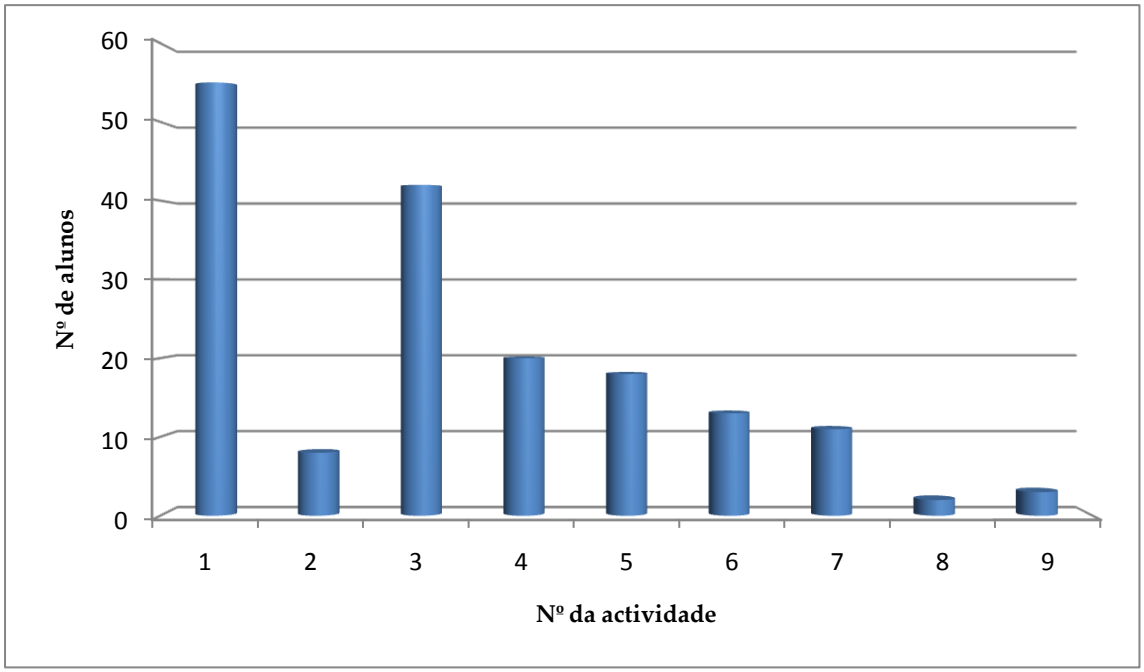


Figura 6.26: Distribuição do número de alunos que realizou cada actividade

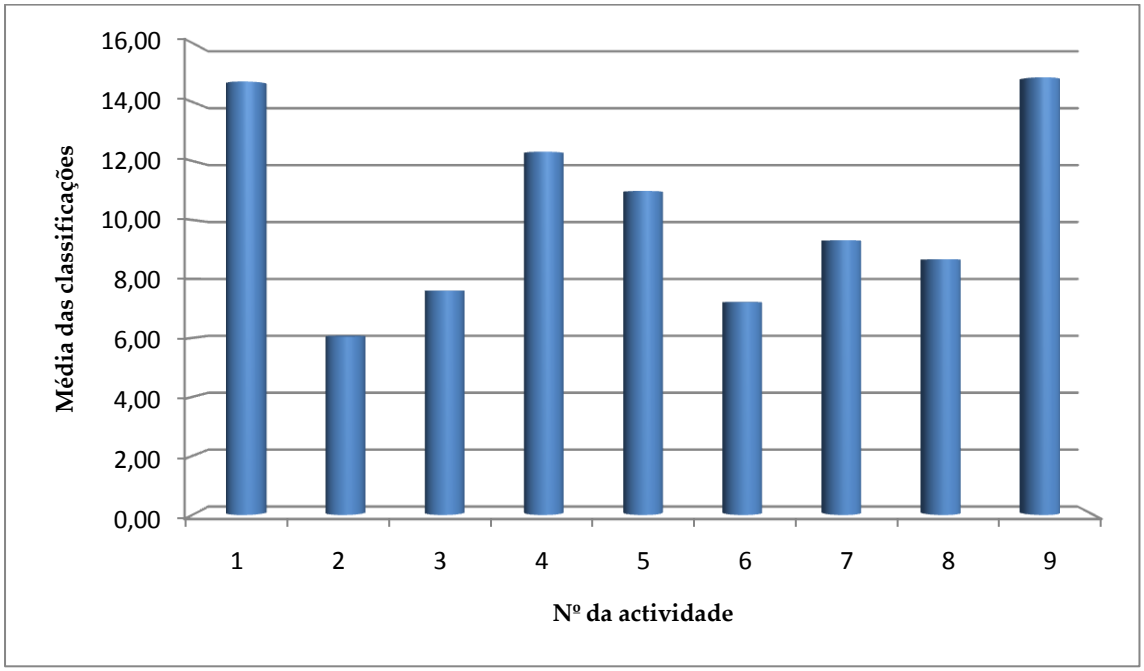


Figura 6.27: Distribuição da média das actividades

De salientar, em geral, o decréscimo de participação, dos alunos, actividade após actividade, havendo apenas uma excepção referente à Actividade 2, para a qual não temos explicação. Este decréscimo reflecte igualmente um decréscimo do número de alunos participantes nas aulas práticas que se traduz numa taxa de desistência alarmante à qual os professores deveriam dar particular importância. Provavelmente, tal facto deve-se à consciencialização por parte dos alunos relativamente à grande dificuldade em realizar com

sucesso a disciplina de PPP depois de terem obtido classificações muito baixas à disciplina sua antecessora, IPRP. De notar também que, apesar de a média das actividades ser relativamente baixa, os alunos que se mantiveram até à última actividade foram os que obtiveram classificações mais elevadas.

Posteriormente, tentou-se perceber a relação entre o número de actividades em que os alunos participaram, bem como os resultados obtidos nessas actividades com cada um dos enfoques do instrumento IACHE. Esta informação pode ser analisada na tabela 6.77.

	Nº de Actividades	**Média das Actividades**
Enfoque Compreensivo	-	-
Enfoque Reprodutivo	-	r=-0,184*
Enfoque Motivação	-	-
Enfoque Percepções Pessoais	r=-0,320**	r=-0,313**
Enfoque Organização	-	-

*(*ao nível 0,05 (2-tailed), **ao nível 0,01 (2-tailed))*

Tabela 6.77: Correlação entre as actividades e os enfoques

Analisando a informação constante na tabela anterior, verifica-se que as percepções pessoais são os aspectos que mais influenciam e condicionam a quantidade e qualidade das tarefas realizadas. Este revelou ser o único enfoque com influência para o número de actividades realizadas. Curiosamente e contrariando as expectativas iniciais, a motivação não revelou ter influência na quantidade e qualidade das actividades realizadas. Esta falta de correlação levou-nos a fazer a separação da amostra comparando os resultados dos alunos que considerámos com níveis diferentes de motivação em dois grupos mas, ainda assim não obtivemos quaisquer resultados de correlação. Talvez o facto de uma pessoa estar fortemente motivada para realizar uma tarefa não seja o suficiente para que se envolva nela. Porém, talvez a percepção de capacidade e autoconfiança potencialize a participação de um indivíduo ou o leve a desistir, caso tenha receio do insucesso.

Outro resultado obtido diz respeito à correlação negativa entre o enfoque reprodutivo e a média dos resultados das actividades realizadas, porém esta correlação já não foi obtida com o número de actividades. Estes resultados significam, na nossa perspectiva, que o mais importante é a qualidade das actividades realizadas e não a sua quantidade. A correlação negativa significa que quanto melhor é o resultado das actividades, menor é o índice reprodutivo. Uma vez que o índice reprodutivo está associado a uma abordagem ao estudo mais superficial e memorística, pensamos que esta correlação fará sentido. Porém, esta justificação também deveria ocorrer, mas em sentido contrário, face ao enfoque compreensivo. Ou seja, também seria expectável que houvesse correlação positiva entre a qualidade das tarefas realizadas e o enfoque compreensivo, significando que os alunos que seguissem uma abordagem de estudo mais profunda tivessem melhores resultados nas

actividades realizadas. Porém, esta correlação não ocorreu. Talvez, a não correlação com o enfoque compreensivo, revele que a profundidade de compreensão será necessária mas não suficiente, só por si, para obter bons resultados nas actividades de programação.

De seguida tentou-se perceber o impacto do número de actividades realizadas bem como os resultados obtidos nessas actividades, nos resultados finais às disciplinas de programação. De notar que estas actividades foram realizadas no contexto da 2ª disciplina de programação (PPP), alvo principal deste estudo. Verificámos a ocorrência de correlação entre a quantidade e qualidade das actividades realizadas e as notas a PPP. Esta correlação significa que os alunos que mais realizaram actividades foram os que obtiveram melhores resultados a PPP. Também os alunos que melhores classificações obtiveram nas actividades realizadas foram os que obtiveram melhores resultados a PPP.

Fomos ainda verificar qual a relação entre os alunos que realizaram actividades bem como a sua qualidade, com os resultados obtidos à primeira disciplina de programação (IPRP). Neste caso obtiveram-se correlações fortíssimas apenas com as classificações médias das actividades realizadas. Isto significa que, os alunos que obtiveram melhores resultados a IPRP, traduzidos pelas classificações finais obtidas nesta disciplina, foram também os que obtiveram melhores resultados nas actividades realizadas em PPP, pois seriam os que melhor programavam. Isto vem reforçar a ideia da grande dificuldade de obter sucesso a PPP por parte dos alunos sem êxito a IPRP. Estes resultados encontram-se na tabela 6.78.

	Nº de Actividades	**Média das Actividades**
IPRP	-	r=0,400**
PPP	r=0,258**	r=0,288*

*(*ao nível 0,05 (2-tailed), **ao nível 0,01 (2-tailed))*

Tabela 6.78: Correlação entre as actividades e os resultados a programação

De seguida analisou-se a relação entre os diferentes enfoques relativos às atitudes e comportamentos de estudo com os resultados obtidos às duas disciplinas de programação (PPP e IPRP). Esta informação encontra-se ilustrada na tabela 6.79.

	IPRP	**PPP**
Enfoque Compreensivo	-	-
Enfoque Reprodutivo	r=-0,192*	r=-0,310**
Motivação	r=0,193*	-
Percepções Pessoais	r=-0,312**	r=-0,348**
Organização	-	-

*(*ao nível 0,05 (2-tailed), **ao nível 0,01 (2-tailed))*

Tabela 6.79: Correlação entre os enfoques e os resultados a programação

De salientar, novamente, a forte correlação obtida entre as percepções pessoais dos alunos e os seus resultados quer nas actividades realizadas quer nas disciplinas de programação envolvidas, nomeadamente IPRP e PPP. Esta análise demonstrou que quanto mais elevadas são as notas nas actividades em que participam e, consequentemente nas disciplinas de programação, mais elevadas são as percepções pessoais dos alunos. Na 1ª disciplina de programação, verificámos que não apenas as percepções pessoais como também a motivação tinham influência para a obtenção de bons resultados. Uma vez que as notas de IPRP e PPP estão fortemente correlacionadas (0,421 ao nível 0,01 *2-tailed*) seria de esperar que o mesmo acontecesse a PPP. Além disso, essa correlação já tinha sido obtida aquando do estudo E, numa outra amostra maior que incluía alguns destes alunos. Porém, apesar de no estudo E, os níveis motivacionais estarem correlacionados com ambas as disciplinas de programação IPRP e PPP, na presente amostra, os níveis motivacionais não se revelaram importantes para o sucesso a PPP. Será que a utilização de uma nova estratégia de ensino, em parte da amostra, utilizando actividades de avaliação formativa de consolidação de conhecimentos, criou um efeito de expectativa anulando o efeito motivacional? Pensamos que a explicação deste fenómeno teria de ter em conta outros estudos, nomeadamente os benefícios que diferentes metodologias de ensino/aprendizagem poderão trazer para a aprendizagem e motivação dos alunos.

A correlação negativa obtida entre o enfoque reprodutivo e os resultados às duas disciplinas de programação não causou surpresa pois aceitamos e concordamos com o facto de as abordagens mais memorísticas e superficiais serem incompatíveis com os desempenhos a disciplinas de programação que exigem estudos profundos e compreensivos. No entanto, esta correlação não foi obtida anteriormente (estudo E), quando considerámos uma amostra maior incluindo os alunos do grupo experimental e de controlo).

Por último, tentámos dar resposta à grande questão deste estudo, comparando os resultados obtidos à disciplina de programação PPP, entre o grupo de alunos submetido às actividades e o grupo de controlo onde essas actividades não foram realizadas. Apesar do número inicial de alunos escolhidos para a participação nessas actividades ser elevado, o facto é que o número de alunos que realizaram todas ou grande parte das actividades foi muito reduzido. Face ao exposto considerámos no grupo experimental apenas os alunos que realizaram pelo menos metade das actividades, ficando este número no limiar do número de elementos admissível para a realização de tratamento estatístico. Porém, não foi obtida qualquer correlação entre os alunos envolvidos na realização das actividades e os resultados a programação (PPP) nem entre os alunos do grupo de controlo e os resultados que obtiveram a PPP. A aplicação do teste de Levene e do teste t entre os grupos levou-nos a

aceitar a hipótese de que não houve diferenças de médias a PPP entre ambos os grupos (experimental e de controlo).

6.7.5 Conclusões

Este estudo não tinha como intuito verificar qualquer questão de investigação principal, no entanto, teve outros objectivos que consideramos igualmente importantes. Particularmente, tencionava avaliar o impacto de uma nova metodologia de ensino, nos resultados finais obtidos a programação. Esta nova metodologia foi traduzida pela realização de actividades no início de todas as aulas práticas/laboratoriais, referentes às matérias leccionadas nas aulas anteriores. A avaliação desta nova metodologia foi traduzida através do estudo de correlações entre a quantidade e qualidade das actividades realizadas com os desempenhos obtidos a uma disciplina de programação. Apesar de os resultados não evidenciarem diferenças estatísticas significativas entre o grupo experimental (sujeito à nova abordagem) e o grupo de controlo, com impacto nos resultados obtidos à disciplina de programação onde essa abordagem foi experimentada, proporcionou resultados que merecem reflexão.

Em primeiro lugar merece destaque a correlação obtida entre o número e a qualidade das actividades realizadas e os desempenhos obtidos à disciplina de programação onde essa abordagem foi aplicada. Outro aspecto que merece reflexão foi o reaparecimento da forte correlação obtida entre as percepções pessoais e os desempenhos obtidos a programação. Por último, a forte correlação negativa obtida entre o enfoque reprodutivo e os desempenhos a programação, significando que quanto mais técnicas de estudo consistindo em abordagens superficiais os alunos utilizam piores são os resultados a programação.

Em geral, este estudo permitiu-nos recolher as conclusões apresentadas na tabela 6.80.

Estudo F - Conclusões	
Enfoque Compreensivo	Médio-alto.
Enfoque Reprodutivo	Médio-alto. Foi encontrada correlação entre o enfoque reprodutivo e a qualidade das actividades realizadas pelo grupo experimental da amostra. Foi encontrada correlação entre o desempenho na primeira disciplina de programação (IPRP) e o enfoque reprodutivo do grupo experimental da amostra. Foi encontrada correlação entre o desempenho na segunda disciplina de programação (PPP) e o enfoque reprodutivo do grupo experimental da amostra.
Enfoque Percepções Pessoais	Médio-Baixo. Foi encontrada correlação forte entre as percepções pessoais e o número de actividades realizadas pelo grupo experimental da amostra. Foi encontrada correlação forte entre as percepções pessoais e a qualidade das actividades realizadas pelo grupo experimental da amostra. Foi encontrada correlação forte entre o desempenho na primeira disciplina de programação (IPRP) e as percepções pessoais do grupo experimental da amostra. Foi encontrada correlação forte entre o desempenho na segunda disciplina de programação (PPP) e as percepções pessoais do grupo experimental da amostra.
Enfoque Motivação	Médio-alto. Foi encontrada correlação entre o desempenho na primeira disciplina de programação (IPRP) e os níveis motivacionais do grupo experimental da amostra.
Enfoque Organização	Médio.
Nº de actividades	Foi encontrada correlação forte entre o desempenho na segunda disciplina de programação (PPP) e o número de actividades realizadas pelo grupo experimental da amostra.
Média das actividades	Foi encontrada correlação entre o desempenho na primeira disciplina de programação (IPRP) e a qualidade das actividades realizadas pelo grupo experimental da amostra. Foi encontrada correlação entre o desempenho na segunda disciplina de programação (PPP) e a qualidade das actividades realizadas pelo grupo experimental da amostra.

Tabela 6.80: Conclusões do Estudo F

6.8 Conclusões gerais

Neste capítulo foram apresentados estudos conduzidos de forma a responder às principais questões de investigação. Os estudos nem sempre foram conduzidos nas situações ideias, mas nas possíveis e muito graças à boa vontade dos professores que connosco colaboraram. Por outro lado os estudos nem sempre se afiguraram fáceis, pois o problema em estudo era complexo, pelo que existia uma multiplicidade de variáveis que por vezes era necessário controlar para não perturbarem os resultados. Desta forma a investigação foi avançando, chamando determinadas variáveis em certos momentos, vigiando-as sempre que possível e, na impossibilidade de controlar tudo, fazendo tentativas de introdução de outras variáveis que em determinados momentos se revelaram mais pertinentes.

Dada a complexidade do problema, foi utilizado o método da triangulação, através da utilização de múltiplos métodos sobre uma série de estudos que ao longo dos tempos foram acumulando alguma evidência. Ao juntar e comparar múltiplas fontes de dados umas com as outras e confrontando os métodos, foi possível corroborar algumas descobertas para a maximização da sua validade (interna e externa), tendo como referência o mesmo problema de investigação.

Também foi utilizada, por vezes, a metodologia qualitativa de Investigação-Acção. Numa primeira aproximação, inicialmente grosseira, planeavam-se as acções a desencadear, posteriormente desencadeavam-se as acções planeadas e finalmente reflectia-se sobre o resultado obtido. Este processo desencadeou, por vezes, novos ciclos de planeamento, acção e reflexão.

Nem sempre foi útil ou possível utilizar metodologias quantitativas. Porém, tal como referido por Fincher e Petre (2004) em *Computer Science Education* a prova é impossível, há apenas evidências fortes ou fracas, pelo que os estudos realizados foram, por vezes, avaliados no contexto de outros estudos que forneceram evidências relevantes. Adicionalmente, o propósito dos estudos realizados não foi o de testar hipóteses ou de apenas observar comportamentos, mas sobretudo em reflectir sobre eles, pelo que por vezes se recorreu à inferência.

Desta forma, os diversos estudos permitiram tecer algumas considerações relativamente aos factores de investigação formulados.

O factor F1 – "Muitos alunos com dificuldades de aprendizagem de programação não possuem as competências necessárias exigidas por uma disciplina introdutória de programação", foi analisado através dos seguintes estudos:

- Estudo A e Estudo B que demonstraram que "Muitos alunos com dificuldades de aprendizagem de programação apresentam muitas dificuldades em resolver problemas" (F1.1).
- Estudo A, Estudo C e parcialmente pelo Estudo B (exceptuando-se as dificuldades de raciocínio lógico) que provaram que "Muitos alunos com dificuldades de aprendizagem de programação apresentam défices de conhecimentos matemáticos e lógicos" (F1.2).

O factor F2 – "As condições e métodos de ensino habitualmente utilizados não são os mais adequados para o ensino de programação", foi analisado através dos seguintes estudos:

- Estudo A, Estudo B e Estudo D que indiciaram ou demonstraram que "A Engenharia Informática/Ciências da Computação atrai alunos com determinado perfil de estilos de aprendizagem" (F2.1).
- Nenhum estudo provou que "Existe correlação entre os estilos de aprendizagem dos alunos e os seus resultados à primeira disciplina de programação" (F2.2). O facto de não se ter obtido correlação, pode significar que a questão é falsa e como tal que não existe correlação ou que o estudo não foi bem conduzido de forma a permitir formar uma conclusão a favor ou contra esta hipótese. Sendo assim, se o factor F2.2 for falso, o que muito nos satisfaria, poderá ter vários significados. Uma das possíveis interpretações é que as estratégias habituais de ensino já contemplam os estilos de aprendizagem dos alunos. Outra possibilidade é a de não ter qualquer influência nos resultados a programação. Alternativamente, poderá significar que os alunos independentemente do seu perfil de estilos de aprendizagem já se terão adaptado às estratégias e estilos de ensino dos seus professores. Também não descuramos o facto de, em geral, os alunos das amostras estudadas apresentarem, na generalidade, preferências fracas ou moderadas por determinada dimensão dos estilos de aprendizagem. Sendo assim, não apresentando preferências fortes poderão não ser afectados por eventuais estilos de ensino que não se enquadrem nas estratégias que mais apreciam. Relativamente à forma como o estudo foi conduzido, pensamos que foi o melhor face às circunstâncias, se bem que uma amostra extremamente grande com a possibilidade de obtenção de números representativos em cada perfil

poderia chegar a conclusões mais realísticas. Alternativamente a aplicações de outro modelo seria um aspecto a não rejeitar.

- Estudo C e Estudo D, que mostraram que "As tarefas habitualmente propostas pelos professores apresentam níveis de dificuldades desajustados ao nível cognitivo do aluno" (F2.3).

O factor F3 – "Os métodos de estudo utilizados pelos alunos não são os mais adequados para a aprendizagem em geral e da programação em particular", não foi provado nem rejeitado, através dos estudos utilizados, nomeadamente Estudo E e Estudo F. Porém, o Estudo F verificou que as abordagens superficiais e memorísticas estão correlacionadas com maus desempenhos a programação.

O factor F4 - "As percepções pessoais dos alunos são em geral muito baixas, insuficientes para enfrentar as exigências das disciplinas de programação", foi analisado através do Estudo E e do Estudo F. Estes estudos mostraram que as percepções pessoais dos alunos estão fortemente relacionadas com os desempenhos obtidos a disciplinas introdutórias de programação, bem como com o seu desempenho em actividades de programação. Assim, nas amostras estudadas, apenas os alunos com as percepções pessoais mais elevadas obtiveram sucesso a programação.

Cap. 7

Proposta de ensino e aprendizagem

"Nada é mais difícil de executar, mais duvidoso de ter êxito ou mais perigoso de manejar do que dar início a uma nova ordem de coisas. O reformador tem inimigos em todos os que lucram com a velha ordem e apenas defensores tépidos nos que lucrariam com a nova ordem."

Nicolau Maquiavel

7.1 Introdução

A literatura analisada e os estudos realizados levam-nos a considerar que é possível interferir em diversos aspectos, no sentido de melhorar o ensino e aprendizagem de programação básica, traduzindo-se em melhores resultados de aprendizagem e consequentemente na redução das taxas de insucesso em disciplinas introdutórias de programação. Apesar de os aspectos analisados revelarem que o problema está principalmente relacionado com três factores, nomeadamente, o conhecimento prévio dos alunos, os métodos de ensino e os métodos de aprendizagem, pensamos ter de ser a instituição e os professores os primeiros impulsionadores desta mudança. É com base neste pressuposto que, neste capítulo, se apresenta uma proposta de ensino e aprendizagem de programação.

7.2 Estrutura lectiva actual

Estamos convictos de que a estrutura lectiva tradicionalmente utilizada para ensinar programação enferma de um conjunto de problemas que poderão condicionar os métodos de

ensino. Estes, por consequência afectarão os resultados de aprendizagem, por não se adequarem às necessidades da maioria dos alunos, em particular para a aprendizagem de programação, por diferentes razões, de seguida apresentadas.

As turmas são demasiado grandes, fazendo com que o ensino não possa ser personalizado, independentemente da vontade do professor, o que leva a que seja muito difícil fornecer, em sala de aula, um acompanhamento e supervisão individualizados adequados e personalizados às necessidades cognitivas de cada aluno. Normalmente, com turmas com muitos alunos, quando o professor chega a conhecer alguns alunos, muitos outros já desistiram e já não comparecem às aulas. Também a diversificação de métodos e estratégias não apenas para contemplar a eventual diversidade de estilos de aprendizagem, mas também para que o professor tenha um melhor conhecimento das dificuldades do aluno e se assegure que cada um adopta a estratégia mais apropriada para o estudo de cada assunto e resolução de cada problema específico, ficam comprometidos. Este factor associado às restrições temporais condiciona a que, mesmo que o professor conheça o benefício de certos métodos pedagógicos, seja levado a utilizar estratégias que por razões práticas se tornam mais fáceis de aplicar, forçando a uma aprendizagem uniforme, obrigando a que todos os alunos aprendam ao mesmo ritmo e de acordo com a estratégia pedagógica definida pelo professor.

A divisão entre aulas teóricas e práticas nuns casos ou teóricas, teórico-práticas e laboratoriais, noutros, apresenta vários problemas. Em termos de aulas teóricas a matéria é habitualmente leccionada de forma demasiado abstracta e mais expositiva, com poucas ou insuficientes representações visuais e/ou animações. O elevado número de alunos nestas aulas dificulta também a consideração das suas características e conhecimentos individuais. Também devido às restrições temporais habitualmente associadas às aulas teóricas, não existe tempo nem condições para a aplicação de métodos mais interactivos essenciais para o entendimento de conceitos de natureza dinâmica, que devem ser experimentados e não apenas demonstrados. Do ponto de vista dos alunos, existe também a ideia de que as aulas teóricas são por natureza aborrecidas, levando muitos deles a não comparecerem, incitando a comportamentos de interrupção e não assiduidade. Sendo nestas aulas que os conceitos básicos de programação são introduzidos, é natural que os alunos não os possuindo ou compreendendo, não sejam capazes de os aplicar, quando é esperado que o façam nas aulas teórico-práticas, práticas ou laboratoriais. Estes comportamentos são o suficiente para que o aluno "apanhando" a matéria em saltos e desgarrada, se desmotive e acabe por desistir ou não tenha condições para um acompanhamento adequado. Do ponto de vista dos professores, torna-se também complicado gerir um conjunto de continuidades, para permitir uma boa fluidez da matéria, nos diferentes tipos de aulas leccionadas habitualmente por

diversos professores. Também consideramos que a distribuição horária por diferentes tipos de aulas leva a um mau aproveitamento do tempo, que poderia ser utilizado para actividades de programação. Por exemplo, nalguns casos as aulas teórico-práticas servem apenas para esclarecimento das dúvidas apresentadas pelos alunos, o que leva a que os alunos frequentemente desistam de comparecer. Assim, em situações que recorram a modelos com por exemplo duas horas semanais atribuídas a cada um dos tipos de aulas, teóricas, teórico-práticas e laboratoriais, acaba-se por aproveitar apenas duas horas semanais para realizar actividades de programação.

A abordagem do ensino não é, muitas vezes, a mais adequada. Nas aulas teóricas 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 perceberem a sua utilidade geral, terem um bom domínio de importantes conceitos de programação ou do que é programar. Consideramos que outra lacuna consiste no facto de se pretender que o aluno comece a programar antes de conhecer o modelo computacional subjacente. Acontece frequentemente que o professor não apresenta de forma suficientemente clara e esclarecedora o modelo computacional bem como o contexto onde uma linguagem de programação se insere, neste modelo. Porque se esquece, porque supõe que isso é feito noutra disciplina ou porque o tempo é insuficiente e tem de se adiantar na matéria para que os professores das turmas teórico-práticas, práticas ou laboratoriais possam também avançar.

Nas aulas práticas as actividades consistem normalmente na realização de programas completos desde a primeira aula, onde adicionalmente os alunos têm de dominar uma ferramenta complexa para aprender a compilar os seus programas. Apesar da abordagem normalmente utilizada nas aulas teóricas dificilmente promover uma verdadeira compreensão de conceitos, para todos os alunos, a abordagem utilizada nas aulas práticas implica a realização de exercícios de níveis cognitivos mais elevados como aplicação, análise e síntese. Este aspecto representa um salto ao qual a maioria dos alunos não consegue responder positivamente. Adicionalmente, a aproximação tradicionalmente utilizada no ensino introdutório de programação consiste em "convidar" os alunos para uma caminhada quase aleatória. Aos estudantes são mostrados alguns programas já construídos, sendo-lhes posteriormente pedido que resolvam programas completos, por si mesmos, muitas vezes

com pouca orientação. No entanto, este tipo de tarefas acarreta uma enorme carga cognitiva e, quando a carga cognitiva do problema a ser resolvido é demasiado elevada, não se pode esperar que haja grande aprendizagem. Este factor, pode levar muitos alunos a desistir logo numa fase inicial, pois são confrontados com uma série de armadilhas e labirintos que rapidamente os fazem perder.

7.3 Estrutura lectiva proposta

Face aos problemas referidos anteriormente relativos à estrutura de ensino actual, propomos uma nova estrutura para o ensino e aprendizagem de programação, de seguida apresentada.

Existência de turmas com um máximo de vinte alunos de forma a que o professor melhor conheça os alunos, e assim possa adaptar um conjunto de aspectos, nomeadamente as estratégias de ensino, o ritmo e tipo de actividades, de forma a serem mais adequadas às dificuldades, preferências de aprendizagem e personalidade dos alunos. Este aspecto poderá ter impacto motivacional não apenas nos alunos como também no professor, ao sentir que o seu esforço está a ter resultados. É claro que esta abordagem implica um investimento em termos de recursos docentes que julgamos possível recuperar a longo prazo com a redução das taxas de insucesso.

Existência de um único tipo de aulas, sem distinção de teóricas, teórico-práticas ou laboratoriais, onde a cada professor é atribuída a responsabilidade de leccionação de todos os tipos de conteúdos. Nessa situação o professor poderá aplicar uma abordagem mais interactiva e menos expositiva, introduzindo os conceitos à medida das necessidades, de acordo com os desempenhos do aluno e o seu nível cognitivo, tendo por base uma taxonomia de objectivos educacionais.

A abordagem do ensino proposta enfatiza a resolução de problemas, sendo os detalhes sintácticos utilizados apenas como veículo para poder programar e relegados para aulas em que já existe um conhecimento mínimo do modelo computacional e do que é programar. Adicionalmente, esta proposta tem como consideração fundamental o aluno, pelo que este poderá e deverá explorar e procurar a informação de acordo com as suas necessidades cognitivas. Sendo assim, o principal referencial é que o ensino e aprendizagem estejam centrados no aluno, de acordo com os princípios do construtivismo. Para os construtivistas, o aluno é um ser activo que participa na construção do seu saber ao invés de assimilar passivamente os conteúdos que um professor transmite nas aulas. É o aluno que reconstrói o conhecimento existente, dando-lhe um novo significado. Apesar desta aproximação sugerir

alguma flexibilidade sugerimos percursos guiados, de acordo com uma metodologia definida, essencialmente para que haja progresso cognitivo e a nova informação seja ancorada nos conceitos existentes na estrutura cognitiva do aluno, possibilitando assim a aprendizagem. Se o armazenamento da informação for feito arbitrariamente, a consequência é a da não integração da nova informação na estrutura cognitiva do aluno, ocorrendo uma aprendizagem mecânica. De forma, a proporcionar ao aluno uma carga cognitiva adequada, mas em simultâneo desafiadora é necessário que o professor prepare percursos personalizados que lhe forneçam a orientação e suporte necessários. Este apoio pode variar no que concerne ao grau de intervenção do professor, podendo existir situações em que o professor faça uma tarefa na sua totalidade, outras em que faça pequenos pedaços da tarefa, outras em que dê "dicas" ou sugestões ocasionais, e outras ainda em que a sua intervenção se limite a dar conselhos sobre o que fazer de seguida ou o melhor estudo a seguir. Qualquer que seja o nível de intervenção do professor, considera-se de extrema importância a permanência dos seguintes aspectos, em qualquer tarefa em que o aluno esteja envolvido:

- Diversificação de métodos de ensino de forma a contemplar, preferencialmente, todos os estilos de aprendizagem existentes.
- Aplicação dos princípios e etapas, recomendados na literatura para uma eficiente resolução de problemas.
- Aconselhamento do aluno sobre as metodologias gerais de estudo adequadas à programação em geral e a determinados tópicos em particular.

É também importante que o professor saiba reconhecer o seu novo papel, passando a ser o orientador da aprendizagem dos seus alunos. Por um lado, deve intervir de forma sensata, apercebendo-se quando é que o aluno precisa da sua ajuda mas não deve interferir na construção do conhecimento que os alunos vão fazendo. É importante que o professor dê espaço e tempo aos alunos para que eles sejam capazes de incorporar novos saberes.

A especificação desta nova proposta de ensino e aprendizagem de programação é de seguida apresentada na figura 7.1, na figura 7.2 e na figura 7.3.

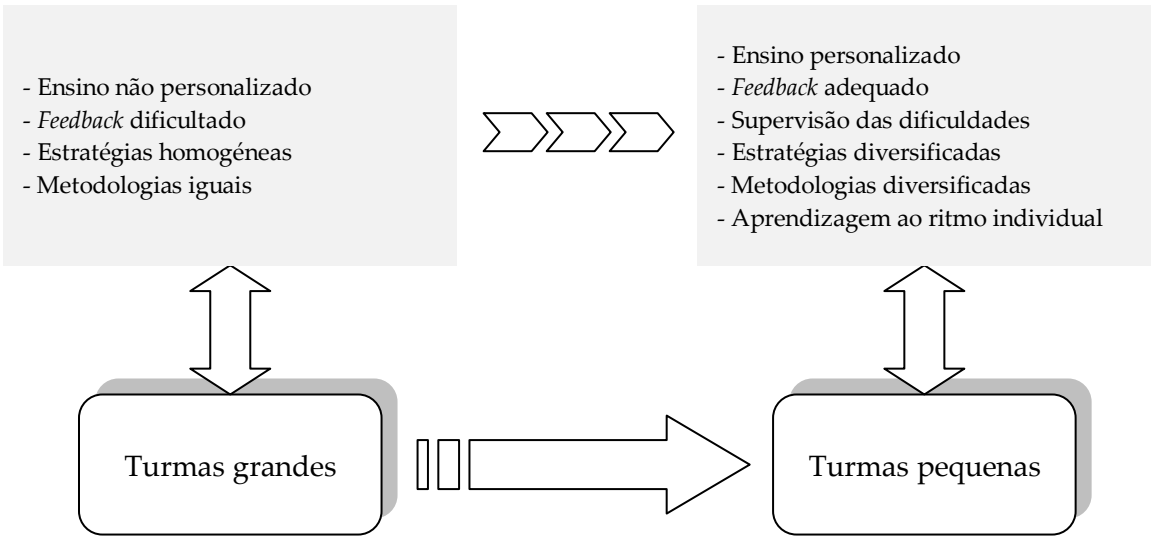


Figura 7.1: Características das turmas – Estrutura actual vs Estrutura proposta.

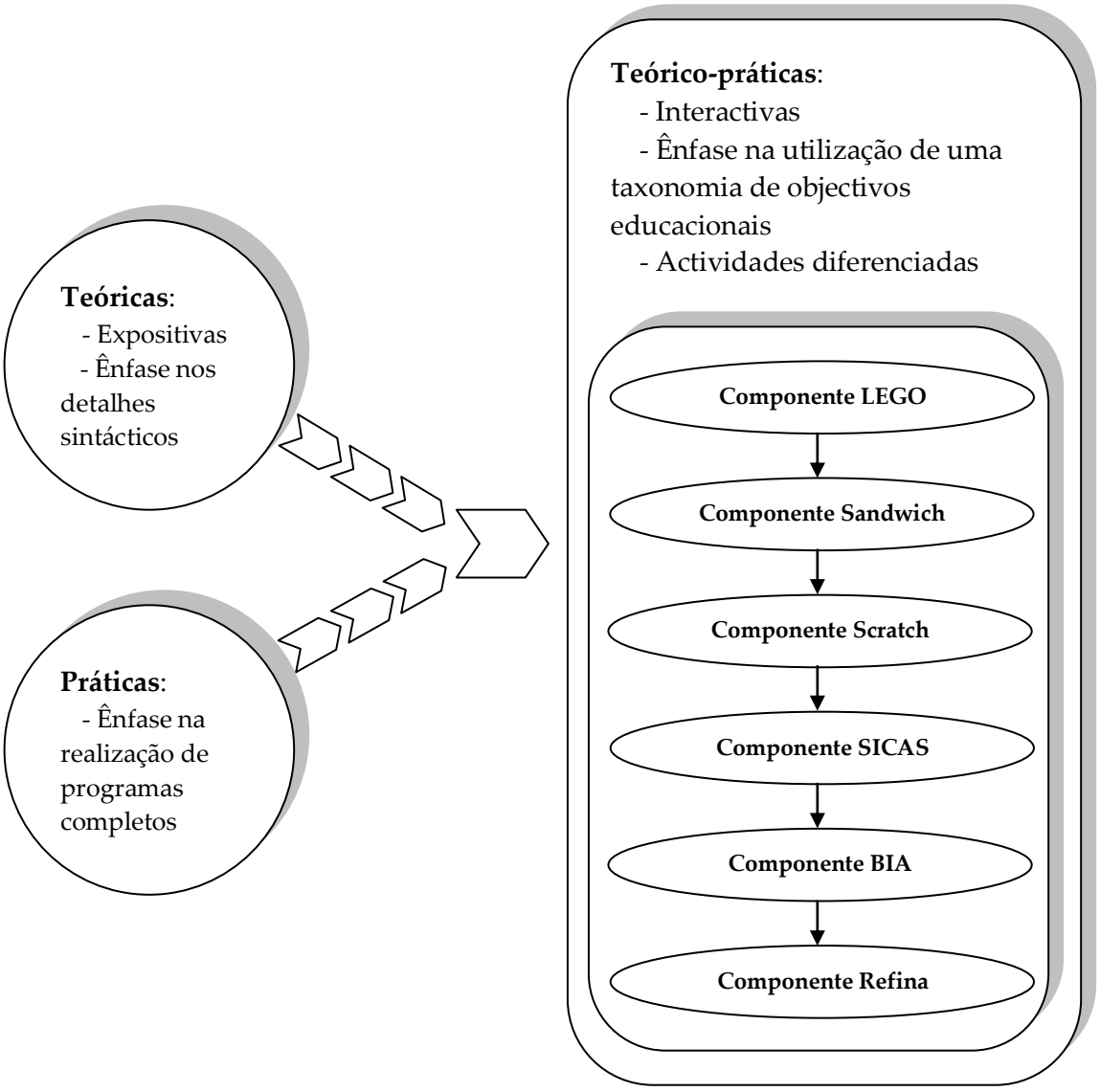


Figura 7.2: Tipo de aulas: Estrutura actual vs Estrutura proposta.

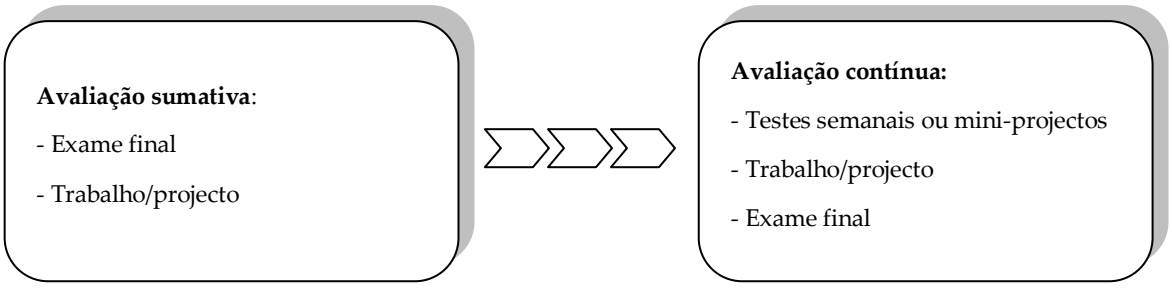


Figura 7.3: Tipo de avaliação: Estrutura actual vs Estrutura proposta.

7.4 Proposta de ensino

Para que os métodos de ensino sejam bem sucedidos, consideramos de primordial importância que o professor conheça as características, preferências e dificuldades dos seus alunos. Consideramos crucial a consciencialização, por parte do professor, da possibilidade de existência de uma variedade de estilos de aprendizagem em cada turma, pelo que, qualquer que seja a metodologia utilizada, esta deverá incluir um esforço no sentido de diversificar as estratégias utilizadas a fim de atender a essa heterogeneidade. Os estudos por nós efectuados, bem como os relatados na literatura evidenciam um padrão de perfis de aprendizagem que caracteriza os alunos das Engenharias, pelo que este poderá ser um útil indicador para o professor. No entanto, se o professor o entender poderá também fazer esse diagnóstico, aplicando, por exemplo, o ILS de Felder e Soloman (Anexo G). Em capítulo dedicado ao assunto (Cap. 3) sugerimos também um conjunto de práticas adequadas para contemplar o ensino e aprendizagem de cada dimensão de estilo de aprendizagem, com o intuito de promover melhores desempenhos e resultados de aprendizagem.

Diagnosticar os alunos que poderão ser alvo iminente de insucesso é também uma preocupação fundamental a que o professor deverá atender. Prever o sucesso dos estudantes em disciplinas introdutórias de programação tem sido uma área de grande pesquisa há mais de 25 anos. Apesar de diversos autores se terem concentrado no desenvolvimento de diferentes instrumentos para prever o sucesso dos alunos nestas disciplinas, até ao momento nenhum revelou resultados significativos e generalizáveis. No entanto, aquando da escolha de qualquer instrumento pensamos ser pertinente ponderar um conjunto de critérios. No caso presente, temos como principal interesse medir algumas das características psicológicas identificadas como sendo importantes para a capacidade de programar, nomeadamente, a capacidade de raciocínio abstracto e matemático e a capacidade de resolver problemas. Porém, como critérios adicionais consideramos importante a sua facilidade e rapidez de administração, o custo do instrumento e a facilidade de análise de resultados e consequente fiabilidade. As pesquisas realizadas não levaram à descoberta de qualquer instrumento com

este conjunto de características. Consideramos que com a estrutura proposta um professor experiente facilmente consiga adquirir um conhecimento das dificuldades de cada aluno com alguma rapidez. Como tal, consideramos que o professor poderá evitar a realização deste tipo de testes e economizar e redireccionar os seus esforços para outros aspectos considerados mais importantes e referidos mais à frente neste documento. Assim, pensamos que estes testes seriam substituíveis pela atenção e proximidade do professor atento a procurar detectar situações críticas desde início. No entanto, caso o professor considere benéfico a existência de um auxiliar para o fazer, pode por exemplo, recorrer ao teste desenvolvido por Dehnadi (2006) (Anexo H). Este teste é inspirado na ideia de que os indivíduos trazem diferentes padrões de conhecimento para qualquer novo processo de aprendizagem. Ao submeterem os alunos a este teste os autores prometem descobrir os modelos mentais dos alunos quando expostos à análise do comportamento de determinado programa e com base na consistência dos modelos identificados preverem a sua capacidade de programação. Porém, este teste apresenta como inconveniente a análise de resultados, baseada na trabalhosa descoberta dos modelos mentais associados, a qual é descrita em Dehnadi (2006). O professor poderá, alternativamente, recolher indicadores sobre as qualificações dos alunos à entrada do ensino superior, em especial no que concerne às classificações obtidas a determinadas disciplinas bem como no que respeita aos seus conhecimentos prévios de programação. Porém, o mais importante será o professor estar vigilante a fim de conhecer os alunos e respectivas dificuldades de forma a poder adaptar as actividades propostas e assim prevenir o insucesso e desistência.

De seguida passa-se a concretizar a proposta de ensino, com base na estrutura sugerida anteriormente, indicando uma metodologia a seguir composta pelos elementos "Componente LEGO", "Componente Sanduíche", "Componente Anima", "Componente BIA" e "Componente Refina".

7.4.1 Componente LEGO

Como consideramos importante que antes de aprender a programar o aluno conheça minimamente o modelo computacional, propomos que na primeira aula de programação seja feita a apresentação deste modelo e respectivo enquadramento das linguagens de programação. Neste sentido, propomos que antes da apresentação desses conceitos de uma forma expositiva, o professor proporcione o envolvimento dos alunos, desde o início, lançando um conjunto de questões sobre essa temática. Assim, o professor permitirá o estabelecimento de diálogos, reflexões e relacionamento de ideias, averiguando em simultâneo o nível aproximado de conhecimentos dos alunos em determinados conceitos

relativos à composição e funcionamento de um computador. De seguida, exemplificam-se algumas perguntas que o professor poderá utilizar, para dar início à conversação.

- O que é *hardware*?
- O que é *software*?
- Como é que o *hardware* comunica com o *software*?
- Conseguem comunicar com alguém que não fale a mesma linguagem? Como?
- O que é uma linguagem computacional?
- Porque é que os informáticos escrevem programas?
- O seu computador já alguma vez teve um comportamento erróneo? Porque será que isso aconteceu?
- Acham que os computadores pensam?

Para dar resposta a estas questões e situar a necessidade de uma linguagem de programação, o professor poderá começar por apresentar o modelo computacional, explicando a importância e função dos seus elementos. Assim, poderá apresentar os elementos genéricos de um computador referindo que a CPU é o "cérebro" do computador. De seguida, poderá encetar explicações utilizando sempre que ache adequado metáforas com exemplos conhecidos dos alunos. Por exemplo, tal como o cérebro humano a CPU processa sinais e envia comandos para as outras partes do "corpo" (computador). Porém, a CPU apenas consegue processar e comandar de acordo com o que o seu projectista delineou – não é capaz de ter um pensamento e raciocínio próprios. A CPU é concebida para processar instruções. Porém, essas instruções têm de ser escritas de uma forma que a CPU as possa compreender. O conjunto de instruções forma aquilo a que chamamos um programa. A CPU executa um programa, através do processamento das suas instruções. Para processar cada instrução realiza os seguintes passos: primeiro, tem de buscar as instruções (*Fetch*), depois tem de as descodificar para as traduzir para uma linguagem que possa compreender (*Decode*) e finalmente executa-as (*Execute*), de forma a realizar uma tarefa específica.

De seguida o professor proporia um exercício para ajudar os alunos a melhor perceber este funcionamento. Sugerimos, por exemplo, a utilização de uma linguagem constituída por peças LEGO de 1x1, ideia inspirada em Hood e Hood (2005). A linguagem LEGO (Anexo I) é especificada usando peças de 1x1 onde a combinação de cores e posições indicam uma acção específica. A ideia da utilização desta linguagem LEGO é a de tornar os conceitos abstractos mais concretos. Sendo assim, após a explicação dessa linguagem, o professor teria condições

para a aplicação prática dos conceitos teóricos apresentados. Assim, uma proposta de actividade poderia consistir na apresentação de uma criação LEGO em que o objectivo seria os alunos, em grupos ou se preferirem individualmente, e com a ajuda necessária e conveniente do professor, comportarem-se como a CPU, que neste caso teria de descodificar cada peça com base na sua cor e posição. Neste exercício (Anexo I – Exercício1) seria enfatizado o ciclo de processamento da CPU (*Fetch-Decode-Execute*). Assim, cada grupo de alunos desempenharia o papel da CPU, tendo de buscar as instruções (uma de cada vez), descodificar cada instrução de acordo com as regras da linguagem e executar a tarefa que lhe corresponderia. Os objectivos desta tarefa seriam uma melhor compreensão, através da concretização, da forma de processamento da CPU. Nomeadamente seria importante que o professor realçasse que a CPU não pensa, que descodificar instruções é um processo tedioso, mas como a CPU não tem emoções não há problema, pode processar milhares delas sem se aborrecer.

O exercício seguinte poderia consistir em pedir aos alunos que construíssem as suas próprias criações (programas), com base nas regras da linguagem LEGO apresentada. Neste exercício (Anexo I – Exercício2) os alunos desempenhariam o papel de programadores, em que utilizando a linguagem LEGO, construiriam um programa para instruir a CPU a realizar uma série de tarefas (um programa). Para tal, teriam que construir uma criação, peça a peça, sendo necessário especificar o tipo de peça, a sua cor e localização na plataforma base. Através da execução de uma série de instruções, seria colocado determinado número de peças na plataforma, de forma a construir a criação pretendida. Sendo assim, esta linguagem permitiria que a CPU traduzisse uma instrução, especificada através de um conjunto de peças, numa tarefa ou tarefas com algum significado. Após a fase de construção, o professor poderia aproveitar a oportunidade para promover o diálogo sobre a importância da sequência de instruções, analisar situações em que a ordem é crítica (ex: peças em cima de outras peças) e outras em que não tem importância (por exemplo, as peças colocadas directamente na plataforma). Este exemplo poderia também ser estendido para demonstrar optimizações, por exemplo, a CPU poderia ter comportamentos mais rápidos, se dentro de cada grupo um aluno se dedicasse à descodificação das peças, outro a buscá-las e outro a descodificar a sua localização. Outro tipo de optimização poderia consistir na identificação de padrões no programa, utilizados para possibilitar a redução do tempo de processamento, útil para explicar as funções aquando da existência de comportamentos repetitivos. De acordo, com os interesses manifestados, poderão ser explicados outros conceitos que se julguem convenientes, como por exemplo, multiprocessamento. O professor poderá também aproveitar para chamar a atenção para o facto de o trabalho de traduzir determinada criação, instrução a instrução, ser repetitivo e tedioso, por estarmos a utilizar uma linguagem de baixo nível. Poder-se-á referir que a CPU compreende a linguagem extremamente simples,

baseada em LEGOS, mas é possível continuar a abstrair através de uma linguagem mais sofisticada, constituindo assim, um trabalho menos aborrecido. Assim, a compreensão destas limitações e a percepção de como expandir essas possibilidades poderia constituir uma óptima oportunidade para fazer a distinção entre uma linguagem de alto nível e uma de baixo nível. Neste sentido, o professor poderia mostrar que é mais simples de entender a descrição para criar construções de coisas concretas, utilizando uma linguagem mais próxima da linguagem "natural", do que a mesma descrição através de um conjunto de instruções utilizando uma linguagem em termos de cor da peça e posicionamento. Desta forma poderia ser introduzida a diferenciação entre uma linguagem de baixo nível e de alto nível, bem como o conceito de compilador. Outro tipo de actividades propostas poderia consistir em um grupo produzir um programa com determinados erros e outro grupo tentar descobri-los. Os grupos teriam não apenas de descobrir os erros como também o seu tipo, se o erro estava no código (erro do programador) ou se era devido a um problema de execução e descodificação das instruções (erro da CPU). Este exemplo teria interesse do ponto de vista da programação e do processamento. Neste caso, o professor poderia também aproveitar para introduzir o tópico relativo à importância do *debugging*. O professor poderia explicar que determinados tipos de erros como combinações de cores de peças inválidas, localização de coordenadas inválidas e especificação de peças que não existem impedem as instruções de serem executadas (erros sintácticos). A identificação dos diferentes tipos de erros poderia proporcionar importantes discussões sobre o assunto.

Para exercício final, o professor poderia propor aos alunos, que pensassem em criações que envolvessem muito mais peças, com muito mais cores e formatos, analisando o número de combinações possíveis, e mostrando a necessidade de instruções mais complexas, envolvendo maior armazenamento e processamento.

Consideramos que as actividades sugeridas nesta componente poderiam ser realizadas noutras disciplinas, como por exemplo disciplinas de arquitecturas de computadores, no entanto, o que pensamos ser importante é que os alunos tenham conhecimento de um conjunto de conceitos básicos que melhor concretizem as tarefas envolvidas na actividade de programação.

7.4.2 Componente Sanduíche

Tendo o aluno uma ideia da constituição de um computador e da importância de uma linguagem de programação, será útil mostrar genericamente algumas características destas linguagens. Uma estratégia interessante para o fazer (adequada para a 2ª aula) é inspirada na abordagem utilizada por Davis e Rebelsky (2007). Estes autores referem que muitos alunos entram em cursos de ciências da computação com poucas noções ou até equívocos sobre o

que tratam em geral e em particular sobre o que é programar. Consequentemente, consideram particularmente importante introduzir a ideia do pensamento algorítmico, o mais cedo possível. Porém, questionaram-se sobre a forma de introduzir o pensamento algorítmico, orientado à programação, a alunos que não tinham ainda as aptidões necessárias para escrever programas. Pensaram então num formato para o conseguir, procurando em simultâneo estabelecer um ambiente positivo e estimulante à turma. Propuseram assim um exercício que consistia em pedir aos alunos que escrevessem as instruções necessárias para fazer uma sanduíche de manteiga de amendoim e geleia:

"Eu estou com fome e quero fazer uma sanduíche de manteiga de amendoim e geleia. Tenho o seguinte equipamento: (1) um frasco da manteiga de amendoim; (2) um frasco da geleia; (3) alguns pratos de papel; (4) alguns guardanapos; (5) duas facas; (6) duas colheres e (7) alguns pensos rápidos".

Após a apresentação do material disponível, o professor avisou que as instruções tinham de ser inequívocas e claras, pois ele era bastante desajeitado e desentendido. Após um período em que os alunos tentaram especificar a solução, individualmente ou em grupo, o professor pediu-lhes que começassem por lhe indicar as instruções, de forma a cumprir o objectivo. Porém, para tornar a experiência memorável e útil, o professor estudou esta situação com todo o cuidado, preparando-se para interpretar o pior possível as instruções que os alunos lhe indicavam. Também o material foi disposto e as acções executadas para que as instruções tivessem maior probabilidade de funcionar mal. Por exemplo, caso os alunos não especificassem o procedimento para pegar na faca, o professor pegava-lhe pela lâmina. Também o pacote de manteiga e o frasco de geleia eram novos (havia a necessidade de tirar o selo antes de tirar a tampa). O professor poderia fazer ainda um conjunto de procedimentos errados, como colocar a colher ou faca lateralmente ao frasco, em vez de pelo topo, entre outros. Adicionalmente, poderia ainda introduzir outros obstáculos como por exemplo, fazer com que a tampa dos frascos fosse de puxar e não de desenroscar, para que o caso mais frequente "rodar a tampa até que ela saia" não funcionasse. A ideia era a de fazer com que os alunos se lembrassem que existe mais de um tipo de tampa e que a instrução possivelmente prevista pelos alunos mais atentos, rodar repetidamente a tampa no sentido contrário ao dos ponteiros do relógio até que ela se separasse do frasco, poderia nunca funcionar e estarmos perante um ciclo infinito. A ideia dos pensos, apesar de poderem não ser necessários era a de sinalizar o perigo inerente a algoritmos incorrectos. Adicionalmente, se o professor quisesse ter reacções mais chocantes poderia ter escondido um pacote de molho de tomate e ao pegar na faca ao contrário simular um corte.

O exercício deveria terminar com a análise do sucedido e identificação dos componentes gerais do algoritmo usados nas instruções indicadas pelos alunos, em particular, seria importante explicar aos estudantes que utilizaram:

- Operações básicas. Os alunos deveriam ficar conscientes que ao escrever um algoritmo, devem saber quais os dados que estão disponíveis para realizar o algoritmo e que operações podem ser aplicadas.

- Nomeação. Os alunos deveriam aprender que é útil nomear coisas, de acordo com nomes significativos, sempre que escrevem algoritmos. Este aspecto poderia ser proporcionado, durante a experiência, quando a instrução fornecida pelos alunos consistisse, por exemplo, em o professor ter de pegar em algo e ter uma mão ocupada ou não conseguir fazer algo com a mão dominante e os alunos serem forçados a descrever "transfira algo para a sua mão não dominante ou faça algo com a sua mão dominante".

- Sequenciação. Os alunos deveriam perceber que por vezes é importante a ordem ou sequência em que as operações são realizadas. Por exemplo, durante a experiência, deveria ser salientada a situação em que a abertura do frasco tivesse de ser feita antes de começar a tentar tirar o recheio, bem como o facto de o pão ter de estar aberto antes de se começar a colocar o recheio.

- Condições. Deveriam ser criadas situações para que os estudantes verificassem a necessidade de procedimentos específicos para casos especiais. Por exemplo, relembre-se a situação relativa ao tipo de frasco e em função disso a devida orientação para a sua abertura, bem como a verificação de situações especiais (frascos novos com selos).

- Repetições. Os estudantes deveriam aperceber-se da necessidade de comportamentos repetitivos. Por exemplo, para determinado tipo de tampa, desenroscá-la até conseguir abrir o frasco. Outro exemplo, poderia consistir na escolha da quantidade de recheio a colocar no pão, normalmente uma colher ou faca cheias não são suficientes, portanto colocar recheio até o pão ficar barrado de determinada forma. Este exemplo também poderia ser aproveitado para chamar a atenção para o perigo dos ciclos infinitos e salientar que nos algoritmos é necessário especificar a condição de paragem.

Em suma, os objectivos do professor, neste exemplo, consistiam em demonstrar o que é programar e que ao fazê-lo é fácil fazer coisas erradas, pelo que é necessário ser o mais específico possível, e não esquecer que poderão existir casos raros. A lição geral mais

importante que os alunos deveriam tirar deste exercício é que, por vezes, parece que o computador percebe completamente ao contrário as instruções que lhe damos, pelo que é necessário especificar tudo com o máximo detalhe e prever situações anormais e invulgares. Com este ou outro exemplo, conhecido dos alunos, o professor deve enfatizar a utilidade da programação, referindo que um programa tem um ciclo de vida e que é suposto que os programadores saibam lidar com todas as suas fases, nomeadamente: Análise e especificação; Concepção da solução; Escrita do código; Validação e Manutenção. O professor deve também chamar a atenção para o facto da importância crucial das duas primeiras fases, antes de qualquer implementação. Deve ainda referir a utilidade de aprender a programar: resolver problemas que tenham soluções algorítmicas, recorrendo para isso a uma linguagem de programação. Ao longo das aulas o professor poderá chamar a atenção para vários aspectos desta actividade, para continuar a ilustrar vários conceitos, através de uma forma significativa. É claro que outros exemplos poderão ser utilizados para ilustrar uma variedade de conceitos utilizados em programação, com ou sem recurso ao computador.

Destaque-se outros autores que usaram a construção de "sanduíches de manteiga de amendoim e geleia" como um exercício de aprendizagem, nomeadamente, Ollis (1995), Lewandowski e Morehead (1998), Elson e Thomas (2006), embora com algumas variações e chamando a atenção para diferentes aspectos ou com diferentes objectivos de aprendizagem. Assim, apesar de outras variações desta proposta, quanto à forma e/ou conteúdo, cremos que a aplicação desta ou de outra sugestão similar poderá ter resultados muito benéficos para os propósitos pretendidos.

7.4.3 Componente Anima

Outro problema referido está relacionado com o facto de tradicionalmente se tentar ensinar conceitos dinâmicos à custa de materiais estáticos. Porém, a nova estrutura proposta, com um melhor aproveitamento dos tempos lectivos, possibilitará a explicação verbal de cada assunto acompanhada pela respectiva animação ou simulação. Desta forma, para a explicação dos principais componentes de uma linguagem de programação, nomeadamente variáveis e estruturas de controlo como selecções e repetições, propomos duas abordagens que permitirão a animação e/ou simulação dos conceitos pretendidos. De forma a promover a inclusão de todos os alunos, propomos, primeiramente uma abordagem de carácter mais lúdico e posteriormente uma abordagem também animada, mas mais próxima de uma linguagem de programação procedimental tradicional. Neste sentido propomos inicialmente, a utilização do Scratch, uma ferramenta desenvolvida no MIT com o intuito de promover capacidades de programação, permitindo construir actividades/programas com níveis de

complexidade e interactividade diversificados, permitindo concretizações autênticas. A ideia desta ferramenta é a de proporcionar um ambiente de aprendizagem activo e estimulante, gerador de motivação, para tornar mais convidativas as primeiras experiências de programação. A importância de ambientes interactivos e jogos para gerar maior envolvimento na tarefa são bem documentados na literatura, veja-se por exemplo (Giguette, 2003; Rajaravivarma, 2005; Bayliss e Strout, 2006), entre outros.

A principal ideia subjacente à utilização do Scratch é a de proporcionar a realização de actividades de complexidade crescente que possibilitem ao professor analisar o nível de desempenho que cada aluno consegue atingir. Este procedimento poderá, por um lado, evitar a realização de testes iniciais para diagnosticar o nível de desenvoltura dos alunos, relativamente à capacidade de resolução de problemas orientados à programação. Por outro lado, poderá envolver o aluno em actividades interessantes motivando-o a querer programar. Com este ambiente, um aluno muito fraco poderá realizar actividades muito simples e os alunos com maiores aptidões para programar poderão chegar a realizar actividades com um nível de sofisticação mais elevado. No Anexo J, encontram-se propostas de actividades apropriadas a todos os alunos, independentemente do seu nível cognitivo e aptidão para programar. De acordo com as necessidades de cada aluno, o professor poderá estipular diferentes períodos de tempo de utilização desta ferramenta, se assim o considerar oportuno. Consideramos, que esta ferramenta poderá ser bastante útil, em especial para utilização extra aulas, por alunos que apresentem mais dificuldades, a fim de permitir desenvolver actividades concretas, que envolvam essencialmente compreensões gerais sobre variáveis, selecções e repetições, para além da promoção de actividades de resolução de problemas.

Apesar desta abordagem poder promover uma compreensão genérica das várias estruturas de controlo, antes da introdução de uma linguagem de programação real, julgamos não ser suficiente. Desta forma consideramos benéfica a utilização de uma aproximação mais gradual a uma linguagem procedimental tradicional. Neste contexto, propomos a utilização do SICAS (Sistema Interactivo para Construção de Algoritmos e sua Simulação). O SICAS (Anexo K), uma ferramenta por nós desenvolvida, permite que o aluno possa construir as suas próprias resoluções de um problema de programação e posteriormente animá-las e testá-las. Pensamos ser crucial, aquando da explicação de determinados conceitos, em especial das estruturas de carácter repetitivo, que esteja disponível uma ferramenta que ajude a uma visualização e compreensão mais profunda da dinâmica envolvida e também mais próxima de uma linguagem de programação.

Sendo assim, sugerimos uma sequência de aulas com recurso à utilização do SICAS. Após uma demonstração das suas potencialidades, também com esta ferramenta o professor

deverá propor problemas com graus de dificuldade adequados. Neste contexto, o professor poderá seguir uma abordagem relativamente tradicional, começando pelos habituais problemas sequenciais como a escrita de volumes, perímetros ou troca de valores entre variáveis. Posteriormente sugerimos que o professor progrida para a apresentação das estruturas de selecção simples e por fim para as estruturas de repetição. Pensamos, numa fase inicial, ser de extrema importância a demonstração pelo professor das potencialidades das ferramentas e da forma de realização das tarefas. De notar que, apesar de sugerirmos as exemplificações, isso não significa que o aluno tenha uma atitude passiva, pois o professor poderá mesmo assim promover comportamentos participativos. Apesar de acharmos que analisar problemas resolvidos não se traduz inevitavelmente num aumento da capacidade de solucionar novos problemas, pensamos que a demonstração por parte do professor e a observação e análise por parte do aluno são aspectos fundamentais nesta fase inicial. Adicionalmente, consideramos até uma excelente oportunidade para envolver a diversidade de alunos existentes, também através de igual diversidade de actividades. Pensamos que pode ser interessante o professor pedir aos alunos propostas de alterações às resoluções existentes ou de resoluções alternativas e rapidamente testar essas novas resoluções. Também a possibilidade de introduzir algoritmos errados, em especial com o tipo de erros lógicos que os alunos habitualmente cometem, e pedir aos alunos que procurem e corrijam esses erros, pode apresentar um alto valor educativo neste contexto. Após a fase de demonstração, o professor deverá, gradualmente, promover condições para que os alunos possam completar programas ou escrever programas muito simples, antes de escreverem programas mais complexos na sua totalidade. Pensamos que, a utilização do SICAS se deva prolongar por um período de tempo mais alargado do que a do Scratch, estimando-se como referência as três semanas. Porém, em qualquer momento que considere oportuno o professor poderá recorrer quer ao SICAS quer ao Scratch, mesmo numa fase mais adiantada da matéria.

Após estas fases iniciais, pensamos que o professor poderá dar início à apresentação dos detalhes sintácticos essenciais da linguagem de programação a estudar, estabelecendo para tal paralelismos relativamente às representações usadas no SICAS. Posteriormente, poderá começar por introduzir todos os detalhes que ache necessários, nomeadamente a diversidade do tipo de variáveis e utilidade de cada uma delas, progredindo no sentido de leccionar todo o programa previsto, mas sempre em contextos relevantes.

7.4.4 Componente BIA

A componente BIA, proposta para a realização de actividades práticas consistirá na existência de tarefas com diversos níveis de desenvolvimento e dificuldade. O conjunto de actividades propostas (Anexo L) integrantes desta componente é exemplificado para a linguagem de programação C, mas poderá contemplar outra linguagem ou paradigma. Em qualquer dos casos, será de utilidade que o aluno já esteja mais direccionado para a linguagem específica pelo que terá de recorrer a um IDE (*Integrated Development Environment*) adequado. De qualquer das formas, consideramos ser da responsabilidade do professor fazer a preparação prévia do ambiente para que o aluno tenha uma entrada o mais suave possível e não se perca com os formalismos habituais inerentes aos ambientes de desenvolvimento. Sugerimos que se prepare esse ambiente, fornecendo *templates* para que os alunos apenas tenham de se concentrar nas actividades fundamentais. Poderá mesmo fornecer-se um conjunto de passos para que os alunos saibam como compilar, executar e fazer o *debug* de um programa. Posteriormente, em fases mais avançadas sugere-se que os professores vão, gradualmente, explicando os detalhes úteis subjacentes a esses processos e ambientes.

Esta estratégia consistirá na realização de tarefas com vários níveis de dificuldade, nomeadamente, **B**ásico**, I**ntermédio e **A**vançado **(BIA)**, onde os exercícios serão apresentados, tendo como referência, a abordagem taxonómica dos objectivos educacionais de Bloom. Tal como proposto por Lister (2000) concordamos que o nível mais elementar (Básico) deverá enfatizar as capacidades de interpretação e compreensão do código, o nível intermédio (Intermédio) deverá incluir a escrita de pequenos fragmentos de código, mas dentro de um contexto bem definido e o último nível (Avançado) já deverá incluir a escrita completa de programas mais complicados, eventualmente associados a reflexões mais elaboradas. As actividades propostas, tendo como referência a taxonomia de Bloom incluirão actividades de Conhecimento e Compreensão no nível Básico, actividades de Aplicação e Análise no nível Intermédio e actividades de Síntese e Avaliação no nível Avançado.

De acordo com a ideia acima defendida, apesar de ser dada liberdade ao aluno no sentido de explorar cada actividade de acordo com os seus interesses, o professor deverá proporcionar alguma supervisão e apresentar percursos de actividades aconselhadas, pelo que sugerimos um conjunto de orientações para que o aluno possa desenvolver as tarefas de programação de acordo com o seu ritmo e necessidades. Assim, uma vez que o professor poderá ter alunos com diferentes ritmos de aprendizagem e uma vez que não será possível acompanhar completamente cada aluno, sugere-se que cada aula seja direccionada para a realização de actividades, incluindo pelo menos dois exercícios de carácter obrigatório em cada nível e dois de carácter de certificação para uma correcta passagem ao nível seguinte. Assim, será o aluno que terá a responsabilidade de testar as suas respostas. Embora o

professor deva ir controlando o progresso geral de cada aluno em sala de aula e esclarecendo as suas dúvidas pontuais, deverá consciencializar o aluno para ter uma atitude pró-activa, no sentido de o responsabilizar pelo seu percurso de aprendizagem. O professor deverá chamar a atenção para a honestidade do aluno consigo próprio, recorrendo a ajudas sempre que necessário e nunca deixando passar uma dificuldade. Assim, se o aluno errar perguntas do nível de Conhecimento, não se visiona outra solução que não seja estudar os conceitos básicos subjacentes. A partir do nível de Conhecimento e de forma a agilizar o processo de controlo de aprendizagem sugerimos, para cada actividade da estratégia BIA, um procedimento mais automatizado. Desta forma, para aferir o conhecimento de cada nível relativamente a determinado tópico ou conceito sugerimos, por exemplo, que o aluno, em cada nível, realize um número de actividades suplementares correspondentes às que errou. Se o aluno acertar nas duas actividades do mesmo nível, poderá passar às actividades do nível seguinte, não invalidando a realização de actividades suplementares, do mesmo nível, extra-aulas. Se o aluno errar todas as perguntas de determinado nível, será benéfico aconselhar-se junto do professor para que lhe sejam sugeridas tarefas de remediação, eventualmente correspondentes ao nível anterior. Assim, a tarefa do professor será a de durante a aula não só esclarecer as dúvidas dos alunos, mas essencialmente aconselhá-los no sentido de procurarem a melhor abordagem ao estudo perante as suas dificuldades. Considera-se que no final de cada aula possa haver um tempo para discussão conjunta, sugestões de estratégias ou chamadas de atenção para os erros mais frequentes ou aspectos que o professor considere pertinente salientar.

O aluno deverá ser alertado para a importância de chegar ao nível mais elevado para cada actividade proposta, pelo que se não o conseguir no período lectivo a isso destinado deverá ter uma atitude que o leve a procurar ajuda (o professor, os colegas, fóruns de discussão, entre outros). Consoante o caso, o professor poderá também sugerir um conjunto de ferramentas de apoio às dificuldades de determinado aluno, para praticar extra-aulas. Como exemplo, pode propor a prática de actividades através de assistentes adequados para a resolução de problemas de programação de que se destacam os *problets*[23] ou o *quizpack*[24]. Consideramos também de extrema importância a colaboração dos alunos com maiores aptidões de programação com os colegas que apresentam mais dificuldades. O professor poderá também lançar fóruns de discussão mediados pelos alunos mais capacitados, responsabilizando um aluno diferente por cada tema discutido ou actividade realizada.

Outra preocupação que o professor deve ter em conta será a de promover e incutir nos alunos o hábito de realização de testes, após cada tarefa realizada. Assim, dependendo do

[23] A sua filosofia é explicada Cap. 5 – Ferramentas de apoio ao ensino e aprendizagem de programação.
[24] Disponível em http://www.sis.pitt.edu/~taler/QuizPACK.html

tipo de tarefa, os testes podem ser fornecidos pelo professor, como parte da especificação da tarefa, os estudantes podem ser convidados a fazê-los como forma de verificação da actividade ou ainda poderá existir uma combinação dos dois. O importante é garantir que existam testes, sempre que o aluno tem de interpretar ou realizar qualquer codificação.

Outra preocupação a ter em conta diz respeito à natureza do problema a resolver e a sua dependência matemática. Quando os problemas têm uma natureza matemática explícita, aconselha-se a inclusão de uma explicação acompanhada da exemplificação do conceito em questão. Excepcionalmente, o professor poderá recomendar ao aluno o estudo de determinado tipo de conceito matemático ou treino de determinado tipo de exercício.

7.4.5 Componente Refina

Na última etapa, em que se espera que os alunos desenvolvam programas de raiz, com um carácter de maior dificuldade ou em que a explicitação dos procedimentos seja mais ambígua, a abordagem poderá ser ligeiramente diferente. A ideia consistirá numa metodologia com aplicação de refinamentos sucessivos, de forma a possibilitar transformações graduais, no sentido de permitir a evolução de soluções parciais e incompletas para soluções totais e completas, de forma a poder implementar gradualmente um "verdadeiro" problema de programação. Pretende-se utilizar o "princípio do refinamento" e o "princípio do desenvolvimento incremental". O princípio do refinamento (Caspersen e Kölling, 2006) – especificar primeiro, implementar depois – é um instrumento bem conhecido e suporta a separação entre o que fazer do que fazer. Primeiramente os alunos deverão pensar no que têm de fazer e só depois no procedimento de como o fazer. O princípio do desenvolvimento incremental (Caspersen e Kölling, 2006) – resolver primeiro um exercício mais simples – é menos conhecido, mas também praticado frequentemente. É uma das muitas competências tácitas dos peritos. Começar por resolver um problema mais simples relativamente ao proposto implica simplificar ou ignorar alguns dos seus requisitos e endereçar somente uma subparte do problema. Gradualmente, vão-se considerando outras partes até que todo o problema original, com todas as suas especificidades, seja resolvido. Este princípio é uma consubstanciação de um dos mantras de Dijkstra (1976), a separação das dificuldades. É uma maneira de dominar a complexidade e um dos nossos primeiros instrumentos do pensamento. A ideia consiste em utilizar um processo de programação incremental reduzindo a tarefa de programação total a um conjunto de tarefas relativamente simples e em alguns casos mesmo triviais, que podem ser tratadas e verificadas uma de cada vez. Esta é essencialmente uma pequena variação do princípio de refinamento por etapas (Wirth, 1971) em que uma tarefa maior é particionada em etapas menores. Durante este processo, um princípio muitas vezes utilizado é o "*Mañana principle*" (Caspersen e Kölling,

2006). De acordo com ele, quando, por exemplo, durante determinada codificação se deseja ter um determinado método ou função, deve-se escrever o código como se esse método existisse. Apenas mais tarde é que se deve tentar implementá-lo. É claro que, inicialmente, os alunos principiantes em programação não terão uma percepção de como fazer essa divisão de tarefas, mas será da responsabilidade do professor ensinar e orientar os alunos sobre o processo de programação, sobre que incrementos fazer e em que ordem. Em anexo (Anexo M) exemplificam-se actividades para a componente Refina.

Consideramos também que, para os alunos com maiores aptidões de programação, possam estar previstos mini-projectos em substituição das actividades de sala de aula contempladas nas estratégias BIA e Refina.

7.5 Estratégia de ensino: Considerações genéricas

Um aspecto basilar nas estratégias propostas será a preocupação em como abranger todos os estilos de aprendizagem. Para todas as componentes apresentadas, no que concerne à exposição dos assuntos conceptuais será fundamental os professores seguirem todas as recomendações apresentadas na secção 3.6 sobre o assunto. Assim, prevemos a sua exploração no sentido de englobar a diversidade de estilos de aprendizagem presentes em sala de aula, através de uma diversificação de acções, nomeadamente:

- Permitir abranger os alunos verbais e os visuais, através da descrição verbal do problema que, sempre que possível, deverá ser complementada por uma ilustração do pretendido ou um diagrama que traduza os dados do problema e os resultados esperados.

- Permitir que os alunos intuitivos especulem, para resolver determinado problema e simultaneamente promover os alunos sensoriais através do fornecimento da informação concreta com os elementos para resolver o problema, de forma prática, permitindo a manipulação dos objectos (por exemplo, na componente "Sanduíche"), se eles assim o entenderem. Dar tempo suficiente para a minúcia dos alunos sensoriais e chamar a atenção da necessidade de detalhar os procedimentos, devido à pressa dos alunos intuitivos. Também, devido à impaciência dos alunos intuitivos perante os detalhes é preciso algum cuidado referente ao dimensionamento da descrição das actividades (perguntas muito grandes podem induzi-los a começar a responder

às perguntas antes de as lerem completamente, levando-os a cometer erros desnecessários). Relativamente às restrições temporais impostas pelos tempos lectivos, sugerimos que o professor conceda aos alunos todo o tempo que necessitarem para a realização dos exercícios propostos para cada actividade. No entanto, caso os alunos não consigam realizar com sucesso os exercícios cognitivamente mais difíceis, estabelecidos para essa aula, deverão realizá-los extra-aulas. Será, contudo, desejável um domínio completo de todos os exercícios propostos antes da aula seguinte.

- Não esquecer também de proporcionar ligações entre as actividades. Por exemplo, na componente "Sanduíche", como se trata de um exercício mais favorável aos alunos sensoriais do que aos intuitivos, para contemplar estes últimos, é extremamente importante a realização de actividades finais para realçar a ligação deste exemplo com a programação.

- Permitir a participação activa de todos os alunos (activos e reflexivos) presentes em sala de aula. Para tal o professor deverá possibilitar a construção de grupos de trabalho para conceberem a solução do problema. Os activos, que normalmente gostam de trabalhar em grupo, construiriam a solução em grupo. Os reflexivos, que preferem pensar sobre o problema, poderiam primeiro fazê-lo individualmente. Dependendo do tipo de actividade, o professor poderá sugerir as situações mais favoráveis para trabalhar em grupo (por exemplo, "Componente Lego", "Componente Sanduíche" ou "Componente Refina") ou trabalhar de forma isolada e reflexiva ("Componente BIA"). Porém, pensamos que qualquer das componentes poderá beneficiar, posteriormente, da abordagem oposta à seguida inicialmente. Assim, o professor deverá destacar a importância de algumas contrariedades, desafiando os alunos para, após a aplicação da estratégia preferencial optar por uma estratégia de contrariedade.

- Promover os globais, através do desafio de soluções criativas ou de optimização, uma vez que a maioria dos exercícios propostos são essencialmente sequenciais. Assim, o professor poderá apelar aos alunos para que pensem em problemas aparentemente diferentes que comportem o mesmo tipo de solução ou convidando-os a pensarem em situações concretas e suas conhecidas onde possam aplicar os conceitos em causa. Por exemplo, o professor poderá perguntar aos alunos que alterações teriam de realizar se as condições do problema fossem alteradas. Exemplificando, para a "Componente Sanduíche", o professor poderia questionar os alunos sobre as alterações a realizar se o objectivo consistisse em fazer várias sanduíches ou se cada sanduíche pudesse ter

> um recheio à escolha de cada aluno ou se houvesse vários sítios onde pudesse estar o pão ou vários tipos de pão. Nas questões mais conceptuais, uma boa estratégia consistirá em o professor apresentar, para cada caso, uma aplicação para um aspecto de interesse, por exemplo, para a implementação de determinados aspectos de jogos computacionais.

Como já referido anteriormente é também importante considerar o aluno como actor central da aprendizagem e não como um mero agente passivo. Este é também um pressuposto central da proposta de Bolonha. Pensamos que as recomendações dadas anteriormente referentes à reformulação dos métodos de ensino, tendo como principal preocupação o aluno, contribuirão para um seu maior envolvimento no estudo. Logo, cremos que, se os professores conseguirem modificar os contextos de aprendizagem, no sentido de serem melhor percepcionados pelos alunos, poder-se-ão conseguir progressos visíveis na aprendizagem. O professor deverá também ter em conta que, para que a aprendizagem seja significativa, o aluno deverá ter uma atitude positiva e favorável, ou seja, tem de estar motivado para acrescentar o que está a aprender ao que já conhece. É desta forma que se modificam as estruturas cognitivas já existentes. Sendo assim, pensamos que, para que um aluno seja capaz de ser auto-ensinante, é imprescindível a motivação e *feedback* do professor, necessárias para que o aluno se envolva profundamente nas tarefas de aprendizagem, mas…

"Se os professores não souberem onde estão as raízes da motivação, como podem desenvolvê-las?" (Oxford e Shearin, 1994)

Consideramos então que o professor deve atender às três fontes da motivação que os alunos apresentam para aprender nomeadamente, os interesses naturais do aprendiz (satisfação intrínseca); o professor/escola/ambiente (recompensas extrínsecas) e o sucesso na tarefa (combinação de satisfação e recompensa) (Fisher, 1990). Neste sentido, há um conjunto de comportamentos genéricos que consideramos importantes. Em primeiro lugar, o professor só conseguirá motivar, se ele próprio estiver motivado, o que implica, acima de tudo, mostrar domínio do assunto (autoconfiança e segurança). Depois o professor deverá mostrar interesse pelos alunos, desenvolvendo o "olho de lince", de forma a perceber, o mais possível, o que se passa na sala de aula. Será também importante que o professor apele à participação, responsabilize os alunos pela sua própria aprendizagem (nos primeiros momentos da aprendizagem orientar os alunos, mas retirar progressivamente essa ajuda). É também fundamental que o professor seja claro relativamente às expectativas que o aluno possa ter: chamando a atenção e transmitindo aos alunos o empenho e esforço necessários que precisam de ter para serem bem sucedidos na disciplina. O professor deverá ter a preocupação permanente de se certificar que o aluno está realmente a perceber os assuntos,

diversificando as formas de apresentar os conceitos e actividades, ao mesmo tempo que demonstra interesse por saber o que pensam ou sabem sobre o assunto. Em termos mais concretos, o trabalho de motivação realizado pelo professor, poderá ser traduzido através de um conjunto de comportamentos, que passamos a exemplificar:

- Procurando exercícios relevantes e interessantes para o público-alvo.
- Tanto quanto possível, dar aos alunos a oportunidade de escolher o tipo de tarefa a realizar ou permitir a escolha de temas pelos alunos.
- Levar os alunos a criar vontade de programar e não a dizer que têm que programar, por exemplo, sendo classificados pela complexidade da escolha que fizeram.
- Permitir alguma flexibilidade no cálculo da nota final ou diferentes modalidades de avaliação.
- Nos trabalhos de grupo, juntar um aluno aparentemente desmotivado com alunos entusiásticos relativamente ao tema.
- Encorajar os alunos, sempre que possível, reconhecendo o esforço, desvalorizando os erros cometidos e destacando a importância da aprendizagem através dos erros.
- Fazer com que os alunos se sintam valorizados por todo o seu progresso e não apenas pela sua nota final. Por exemplo, propor aos alunos um problema que em determinado momento não conseguiam resolver ou uma pequena variação deste, mostrando-lhes a sua evolução. De salientar, contudo, a importância de realçar não apenas as qualidades, mas também sendo específico relativamente às áreas a melhorar.
- Ajudar os alunos a reflectir sobre o seu próprio processo de aprendizagem: como e quais os processos que os ajudaram a alcançar determinado conhecimento.
- Estabelecer relações de proximidade com os alunos, predispondo-os para a exibição de comportamentos mais descontraídos, permitindo-lhes assim expor com naturalidade todas as suas dúvidas. Adicionalmente, fazer um esforço para que o aluno encare o professor como alguém que o ajuda a ser bem sucedido e não como alguém que o impede de obter boas notas.

Complementarmente, consideramos que o ensino de estratégias de aprendizagem explícitas sobre determinado conteúdo específico, poderá ser um factor determinante para

que os alunos apresentem uma aprendizagem proficiente nesse domínio específico. Desta forma, a par do trabalho de motivação, o professor, deverá ser conhecedor do caminho mais eficiente e eficaz para o estudo de determinado assunto, dando-o a conhecer aos alunos. Será desejável que o professor sugira estratégias de aprendizagem específicas e integradas no contexto da disciplina, de forma a aconselhar a melhor forma de estudar determinado assunto particular. Caso isso não aconteça, o aluno deve ter a coragem de perguntar ao professor a melhor forma de o fazer.

Porém, a mudança não poderá ocorrer unicamente com o professor mas também com o aluno. É necessária a consciência, por parte do aluno, para que este tenha uma atitude activa e pró-activa da aprendizagem. De acordo com Zimmerman (1989), para maximizar o desempenho académico, é necessário potenciar e actualizar a capacidade do aluno para aprender, a partir da coordenação das aptidões cognitivas, metacognitivas e motivacionais, por serem cruciais no processo de aprendizagem, a par do conhecimento dos conteúdos. No entanto, a investigação tem revelado que, independentemente do nível de estudos, os estudantes revelam défices de competências necessárias para a sua abordagem bem sucedida, não sabendo estudar autonomamente. Neste sentido é fundamental que os alunos desenvolvam a capacidade para regular a sua aprendizagem, desenvolvendo assim uma aprendizagem autónoma e auto-regulada. Um dos componentes com enorme influência para uma aprendizagem auto-regulada é a motivação. De acordo com Bandura (1995), a vertente motivacional inclui a percepção da auto-eficácia, automonitorização, estabelecimento de objectivos e auto-incentivos afectivos bem como as auto-atribuições que implicam esforço, persistência e crenças positivas de competência (Zimmerman, 1990).

Embora o professor possa contribuir grandemente no sentido de orientar o aluno sobre as melhores estratégias para estudar determinada disciplina em geral e cada assunto dessa disciplina em particular, pensamos que as instituições deveriam investir em apoios extra para alunos que os professores possam identificar com dificuldades de aprendizagem. Pensamos que este seria um investimento não apenas para as disciplinas de programação mas para a generalidade das disciplinas. Sendo assim, as instituições deveriam promover a existência de programas com profissionais habilitados com os seguintes objectivos gerais: diagnosticar e estimular as variáveis motivacionais e de autoconceito; promover estratégias de estudo genéricas e profundas; promover programas de treino cognitivo específico. Nestes programas, consideramos de primordial importância, em primeiro lugar, a realização de testes para diagnóstico dos níveis motivacionais e testes para diagnóstico de autoconceito. Isto porque uma auto-estima, autoconceito e expectativas de eficácia pouco positivas irão ter influência na motivação, entusiasmo e persistência do aluno na realização das tarefas e consequentemente nos resultados da aprendizagem. Em qualquer das situações, a motivação

é um factor que não pode, de forma alguma, ser escamoteado. É fundamental perceber os sentimentos e imagens dos alunos em relação às suas capacidades, à sua realização cognitiva e à sua aprendizagem. Uma vez que os níveis motivacionais estão normalmente relacionados com a forma como os alunos se envolvem nas tarefas de aprendizagem, é necessário investir em treinos motivacionais de forma a conduzir a uma abordagem à aprendizagem profunda, tão necessária em disciplinas de programação. A identificação de problemas desta ordem permitiria, posteriormente, realizar testes que proporcionassem mudanças no âmbito das expectativas e das percepções pessoais de competência (Almeida e Balão, 1996 citado em Alves et al., 2002). Após o diagnóstico considera-se pertinente investir no treino motivacional e de auto-eficácia com incidência particular nas funções em défice.

Porém, estamos convictos de que apesar da importância fulcral da motivação e autoconceito, estes elementos, só por si, não serão suficientes para conseguir uma aprendizagem de sucesso. Pensamos que, actualmente, dada a massificação do ensino, a maioria significativa dos alunos, precisa de ajuda urgente, no sentido de serem instruídos em processos e estratégias de uma aprendizagem profunda. Eventualmente possuirão várias dimensões que precisam de ser corrigidas e orientadas, nomeadamente, a sua motivação para aprender, métodos de estudo e estratégias de aprendizagem, resultados de aprendizagem pretendidos, recursos sociais e ambientais, entre outros.

De extrema importância será a integração, nesse programa, de sessões destinadas à promoção de aquisição de competências de estudo. Para tal, poder-se-ão utilizar ferramentas adequadas a cada situação particular que esses profissionais certamente bem conhecerão. Outra sugestão para este efeito é a aplicação do projecto "Cartas de Gervásio ao seu umbigo" (Rosário et al., 2006). O Gervásio, um estudante universitário do 1º ano, elabora um conjunto de cartas, onde, num registo intimista, descreve as suas experiências e reflexões sobre os processos de aprendizagem no contexto académico. Cada carta funciona como que um guião para a auto-regulação, servindo cada uma para apresentar um conjunto de estratégias de aprendizagem relativas a cada fase do processo de auto-regulação (exemplo: estabelecimento de objectivos, organização do tempo, tomada de apontamentos, lidar com a ansiedade face aos exames, entre outros). Pensamos que o ponto mais forte deste projecto, reside na utilização de um personagem com uma vida semelhante e enfrentando os mesmos problemas da maioria dos alunos alvo de insucesso. Para que a aprendizagem seja efectiva é fundamental que os indivíduos percebam as semelhanças entre determinado modelo e a sua situação particular (Rosário et al., 2006).

No projecto, apresentado em livro, são propostas actividades, a propósito de cada carta, organizadas para que os alunos sejam instruídos no processo auto-regulatório, aprendam e treinem estratégias de aprendizagem em diferentes domínios, desenvolvendo

hábitos de reflexão, atitudes críticas ou capacidades de resolução de problemas. O próprio projecto não indica o número de sessões previstas, nem o tempo destinado a cada sessão, sendo apresentado num formato de justaposição curricular. No entanto, é sugerida a contemplação da análise da totalidade das cartas, pelo número de sessões consideradas adequadas, exploradas preferencialmente por um psicólogo ou profissional em estratégias de estudo. As sugestões de trabalho, alocadas a cada carta, são também diversificadas de forma a responder às possíveis necessidades dos diferentes alunos.

No entanto, um aluno poderá ainda revelar outro tipo de défices de aprendizagem, devido a determinadas dificuldades cognitivas. Segundo a abordagem cognitivista as funções cognitivas dividem-se em: atenção; (des)codificação; organização e classificação; retenção e evocação; categorização; inferência e dedução; fluência e flexibilidade e avaliação. Uma dificuldade na aprendizagem pode ser devida a um problema específico em qualquer uma destas funções, pelo que há que detectá-lo e posteriormente apostar no seu treino. Os profissionais referidos deverão também prever a realização de testes que permitam este diagnóstico e respectivo treino. A abordagem do treino cognitivo é uma abordagem do processamento da informação com a finalidade de compreender as capacidades mentais, de um ponto de vista de exercitação dos processos cognitivos. Existem programas que treinam as funções cognitivas básicas e superiores e outros que delimitam o campo de actuação a processos cognitivos mais específicos. Em função do diagnóstico obtido dever-se-á escolher um programa de treino cognitivo adequado. De realçar ainda que a escolha dos programas cognitivos deve ser feita com algumas cautelas, por exemplo, havendo o cuidado de os problemas que os integram estarem relacionados com o quotidiano dos alunos e com os seus interesses: experiências da família e amigos, televisão, cinema, música, entre outros. Os alunos compreendem melhor e conseguem recordar com mais facilidade quando conseguem ver relações entre a matéria estudada e as próprias vivências. Para que este programa seja eficaz, deve possibilitar a simulação de problemas e situações reais do dia-a-dia, a representação das perspectivas de outros e ser um espaço seguro e controlado, que estimule o pensamento do aluno. É importante que incentive a reflexão, através da articulação e representação dos saberes do aluno, através do pensamento sobre os processos que utilizaram e de como chegaram a determinado resultado, estimulando a construção de significados e desenvolvendo o pensamento cognitivo (Jonassen et al., 1999).

Deve também ser tido em consideração que as ferramentas cognitivas apenas são úteis quando apoiam determinado tipo de raciocínio, que se encontra na zona proximal de desenvolvimento do indivíduo, ou seja, a zona entre as capacidades de desenvolvimento existentes e as capacidades potenciais dos alunos. Assim, as ferramentas cognitivas deverão representar "andaimes" cognitivos e, ao observar a zona proximal do aluno, o profissional

deve orientá-lo no sentido de desenvolver o seu potencial, tornando-o real. Desta forma, devem ter-se em consideração as potencialidades, interesses e dificuldades dos vários alunos (tanto a nível cognitivo, como a nível sócio-relacional) e, na medida do possível, respeitar-se os ritmos e preferências de aprendizagem de cada um. É igualmente importante que as condições em que ocorre a aplicação do programa permitam um ambiente descontraído, estimulante e colaborativo. Porém, é muito importante o aluno consciencializar-se que o treino cognitivo passa também pelo seu esforço e persistência e por mudanças nas suas percepções pessoais de capacidade e realização.

Contudo, a exercitação dos alunos no uso de estratégias para ultrapassar as dificuldades detectadas no processamento cognitivo da informação, pode ser improfícuo se não for fomentada e estabelecida a aplicação dessas competências ao contexto específico de aprendizagem do aluno (Ackerman, 1993; Biggs, 1993; Rosário, 1997; Rosário, 1999). Neste contexto, prevemos que o programa de treino poderá ter que ter a intervenção do professor de programação. Por exemplo, os alunos com mais dificuldades a programação, necessitarão eventualmente de um programa de treino centrado num domínio específico de resolução de problemas e algoritmia. Consideramos que a proposta deste programa extra possa ter uma duração variável, mas será conveniente a sua disponibilização ao longo de todo o ano lectivo. Estamos convictos que será necessário um acompanhamento permanente e cíclico de forma a rever constantemente os comportamentos, motivações e controlo da eficácia dos métodos de estudo e dificuldades dos alunos. Neste aspecto, o papel do professor de programação será também de primordial importância, não apenas como orientador de conteúdos a trabalhar mas também como gerador de estímulos. Assim, sempre que possível, o professor deverá trabalhar as motivações dos alunos contribuindo para o melhoramento das suas percepções pessoais.

7.6 Avaliação

Consideramos que uma proposta de ensino não fica completa sem a definição do correspondente método de avaliação. Cremos nas vantagens pedagógicas de uma avaliação contínua e não numa avaliação com um único teste no final da unidade curricular. No entanto, a avaliação contínua também apresenta desvantagens. Por um lado, os gastos temporais retirados às aulas onde normalmente se dispõe de pouco tempo para leccionar todos os tópicos e desenvolver as estratégias pretendidas. Por outro lado, o enorme dispêndio de tempo por parte dos professores para realizar os enunciados das avaliações e outras actividades e respectiva correcção. No entanto, pensamos que um professor competente e interessado pelo sucesso dos seus alunos facilmente constituirá esse tipo de

avaliações, nem que essas impliquem uma cadência semanal. Em termos de correcções propomos que as actividades sejam feitas, por exemplo, através de um sistema de avaliação automática como por exemplo o Mooshak. Alternativamente podemos usar uma plataforma de e-learning, com a modalidade de correcção de testes. É claro que esta abordagem implica que o tipo de actividades seja de escolha múltipla. Porém, os exames de múltipla escolha, são muitas vezes criticados por serem respondidos através de trabalho de adivinhação. Além disso, este tipo de testes aparece frequentemente descrito como a técnica de avaliação de último recurso, sendo conotado como o refúgio dos professores preguiçosos para fazer correcções. Este é um tipo de teste que não é habitual em disciplinas de programação, embora possa ser vantajoso para a avaliação de determinados níveis de conhecimento. Adicionalmente, as vantagens dos testes de escolha múltipla, são muitas, nomeadamente no que concerne à objectividade na correcção para além de poderem ser de grande utilidade pedagógica, o que também é corroborado pela literatura (Ebel e Frisbie, 1986; Linn e Gronlund, 1995; Haladyna, 1999). É claro que esta solução, será trabalhosa mas, facilmente o professor ficará munido de um conjunto de questões que poderá reutilizar ou facilmente adaptar futuramente. Em Traynor e Gibson (2005) e Pillay (2002) é possível encontrar sugestões de técnicas para facilmente produzir bons testes de escolha múltipla sobre códigos de programas. Porém, o maior desafio consistirá em o professor redigir as respostas, utilizando bons distractores, cada um dos quais avaliando um determinado tipo de problema ou não compreensão, relativamente ao tópico e nível que se pretenda avaliar. Um bom distractor deverá ser suficientemente convincente para apelar a alguns estudantes. Porém, deverá ser claramente incorrecto para não induzir em erro os alunos que realmente sabem a matéria. Na prática este é um tipo de tarefa que nem sempre é fácil de realizar.

Propomos então a realização dos seguintes tipos e elementos de avaliação. Avaliação contínua, realizada ao longo das aulas com testes periódicos, predominantemente de escolha múltipla, preferencialmente um semanal, com duração aproximada de quinze minutos. Estes testes/actividades incidiriam sobre a matéria exercitada nas aulas anteriores. Este tipo de testes não teria grande importância em termos de classificação final, mas o mais importante seria disciplinar os alunos para um acompanhamento permanente da matéria e, eventualmente, um maior envolvimento dos alunos e diagnóstico, se possível, dos casos mais problemáticos. Porém, continuamos a considerar importante a realização de um exame individual final.

Nas actividades semanais de avaliação contínua consideramos importante a existência de exercícios que contemplem todos os níveis da taxonomia de Bloom, de forma a melhor identificar as dificuldades dos alunos. Porém, no exame final, pensamos que o nível de exigência deva ser superior implicando, por exemplo, a realização da totalidade de um

programa (de dificuldade Básica); a interpretação de um programa completo (de dificuldade Intermédia) e a realização de um programa completo (de dificuldade Avançada). Consideramos também oportuno que nos momentos de transição da utilização dos ambientes/ferramentas propostas se realizem testes ou actividades de avaliação de conhecimentos das ferramentas ou linguagens estudadas. A informação recolhida servirá essencialmente para o professor poder aconselhar os percursos mais adequados para os alunos colmatarem as dificuldades apresentadas.

Consideramos também, de grande utilidade a existência de um projecto, realizado extra aulas, implicando o desenvolvimento de um trabalho maior, integrador de todos os tópicos leccionados. Consideramos importante que a cotação atribuída a este projecto seja definida em termos das funcionalidades desenvolvidas, de forma a gerar nos alunos a motivação necessária para querer programar mais, tendo a recompensa da classificação.

7.7 Conclusões

As elevadas taxas de reprovação a disciplinas de programação e os estudos realizados levam-nos a considerar que é possível intervir em vários factores, no sentido de melhorar o ensino e aprendizagem de programação. É com base neste pressuposto que, neste capítulo, apresentámos uma proposta de ensino e aprendizagem de programação. Esta proposta é constituída por várias componentes que pretendem ampliar o funcionamento cognitivo do aluno. Em geral, baseia-se na inclusão de actividades diferenciadas para contemplar os diferentes níveis cognitivos, através de uma abordagem interactiva e experimental e menos expositiva e demonstrativa. Outro aspecto chave desta abordagem consiste em centrar-se primeiramente na resolução de problemas de programação em vez de nas características sintácticas de uma linguagem de programação. Adicionalmente, caracteriza-se pela utilização de ferramentas que permitam a concretização de muitas abstracções envolvidas em programação. Estamos também convictos de que esta abordagem permitirá uma maior aproximação entre aluno e professor, estabelecendo relacionamentos mais próximos, envolvendo e motivando uns e outros.

A abordagem proposta, embora pensada especialmente para alunos "menos eficientes", também contempla os alunos mais proficientes. Porém, também não é expectável, que a aplicação desta proposta permita transformar qualquer aluno num programador excepcional. Porém, estamos convictos de que a aplicação de uma abordagem de ensino/aprendizagem motivadora, em que as tarefas são propostas de acordo com um caminho aconselhado, onde os níveis de dificuldade são graduais, permitirá que todos os

alunos conseguirão aprender a programar, desde que se decidam a fazê-lo. Esta decisão de aprender, depende grandemente dos seus níveis motivacionais inter-dependentes da sua capacidade para auto-regular a sua aprendizagem e comportamentos associados, pelo que também apresentámos propostas neste domínio.

A proposta apresentada ainda não foi aplicada, por não ter havido condições para tal, pelo que o passo seguinte consistirá em mobilizar as instituições e professores para a pertinência da sua aplicação. É importante reconhecer que esta abordagem implicará a existência de mais professores com responsabilidades de leccionações teóricas em simultâneo com as leccionações teórico-práticas, práticas e laboratoriais. É claro que esta abordagem terá algumas implicações em termos de recursos, nomeadamente, a existência de mais professores e mais turmas, de tamanho necessariamente mais reduzido, mas estamos convencidos que se traduzirá num grande investimento a médio e longo prazo.

Cap. 8

Conclusões e trabalho futuro

"Não fiz o melhor, mas fiz tudo para que o melhor fosse feito. Não sou o que deveria ser, mas não sou o que era antes."

Marthin Luther King

Os elevados níveis de insucesso nas disciplinas de programação, em qualquer grau e sistema de ensino, em qualquer parte do mundo, indiciam que as práticas de ensino e de aprendizagem deste tipo de assuntos requerem cuidados especiais. Esta preocupação tem sido traduzida, ao longo dos tempos, pelo desenvolvimento de numerosos sistemas e propostas, sem que contudo se tenham verificado diferenças significativas nos resultados de aprendizagem.

Uma percentagem considerável da literatura aponta diversos factores como estando na origem deste problema, sendo as principais causas referentes a défices de pré-requisitos por parte dos alunos. São apontadas principalmente as deficiências apresentadas em termos de capacidade de resolução de problemas, as bases matemáticas inexistentes e a incapacidade de abstracção dos alunos. Na literatura, também surgem opiniões sobre o melhor paradigma para começar a programar, porém, a generalidade das opiniões é a de que independentemente do paradigma ou linguagem escolhida, aprender a programar é uma tarefa cognitivamente exigente, de natureza abstracta, que requer esforço e perseverança, para além de necessitar de uma hierarquia de competências base.

Apesar de menos usual também se encontram referências alusivas ao facto de as disciplinas de programação estarem mal localizadas no currículo, por coincidirem com um período de alguma instabilidade na vida do aluno. Por todos os motivos referidos, é

necessário que um aluno esteja altamente motivado para enfrentar uma disciplina difícil, exigente e que adicionalmente é frequentemente conotada de forma negativa.

8.1 Principais contributos

O objectivo deste trabalho foi o de averiguar as principais dificuldades inerentes à aprendizagem inicial de programação, tentando compreender as razões subjacentes a este problema e procurar soluções que contribuam para o conseguir ultrapassar ou pelo menos minorar. Porém, face à complexidade do problema, o nosso propósito foi o de apreender os factores mais importantes associados à prática do ensino e aprendizagem de programação, pelo que os estudos realizados tiveram como base os problemas referidos na literatura que considerámos pertinentes. De seguida são apresentados os principais contributos deste trabalho nomeadamente no que concerne aos estudos realizados, que nos permitiram responder aos factores de investigação formulados, bem como à apresentação de uma proposta pedagógica de ensino/aprendizagem de programação.

8.1.1 Respostas aos factores de investigação

Os estudos foram dimensionados de forma a analisar três elementos principais, nomeadamente, o conhecimento de base dos alunos, os métodos de ensino e os métodos de aprendizagem.

Relativamente ao conhecimento de base dos alunos considerámos os seguintes factores de investigação:

F1 – Muitos alunos com dificuldades de aprendizagem de programação não possuem as competências necessárias, exigidas por uma disciplina introdutória de programação.

F1.1 – Muitos alunos com dificuldades de aprendizagem de programação apresentam muitas dificuldades em resolver problemas.

F1.2 – Muitos alunos com dificuldades de aprendizagem de programação apresentam défices de conhecimentos matemáticos e lógicos.

Relativamente aos métodos de ensino e ao seu principal actor (professor), considerámos os seguintes factores de investigação:

F2 – As condições e métodos de ensino habitualmente utilizados não são os mais adequados para o ensino de programação.

F2.1 – A Engenharia Informática/Ciências da Computação atrai alunos com determinado perfil de estilos de aprendizagem.

F2.2 – Existe correlação entre os estilos de aprendizagem dos alunos e os seus resultados à primeira disciplina de programação.

F2.3 – As tarefas habitualmente propostas pelos professores apresentam níveis de dificuldades desajustados ao nível cognitivo do aluno.

Relativamente aos métodos de estudo considerámos os seguintes factores de investigação:

F3 – Os métodos de estudo utilizados pelos alunos não são os mais adequados para a aprendizagem em geral e da programação em particular.

F4 – As percepções pessoais dos alunos são em geral muito baixas, insuficientes para enfrentar as exigências das disciplinas de programação.

Relativamente ao factor F1, os estudos realizados demonstraram que os alunos com dificuldades a programação apresentam enormes dificuldades em resolver problemas em geral e em especial aqueles que apresentam uma natureza matemática implícita ou explícita. Os défices de conhecimentos matemáticos foram evidentes entre os alunos estudados, nomeadamente: as dificuldades em traduzir a descrição textual de um problema para uma fórmula matemática que o resolva; a falta de conceitos matemáticos básicos; as dificuldades de cálculo; o desconhecimento de figuras geométricas; as dificuldades de interpretação do enunciado; a incapacidade de definir critérios de comparação; a incapacidade de orientação no plano cartesiano; o desconhecimento de conceitos trigonométricos ou a incapacidade de aplicar conceitos trigonométricos. De todos os aspectos referidos sobressaem os fracos níveis de abstracção, traduzidos pela incapacidade de generalizar situações através de fórmulas ou descrições que resolvam problemas. Apesar de também ser detectada alguma falta de raciocínio lógico, constatou-se que este aspecto se consegue treinar com alguma facilidade.

Relativamente ao factor F2, demonstrou-se que a Engenharia Informática/Ciências da Computação atrai alunos com determinado perfil de estilos de aprendizagem. No entanto não foi encontrada correlação entre os estilos de aprendizagem dos alunos e os seus resultados a programação. O facto dos estudos realizados não evidenciarem diferenças de desempenho a programação relativamente aos diferentes perfis de estilos de aprendizagem, poderá ter diferentes interpretações. Possivelmente, o modelo seguido necessitaria de uma amostra muito maior para revelar essa correlação, dada a diversidade de perfis existentes possibilitado pelo modelo utilizado. Porém, julgamos que ou a maioria dos professores já aplica estratégias favoráveis, pelo menos à maioria dos alunos presentes em sala de aula ou

então que os alunos tenham ultrapassado as situações de aprendizagem que lhes são menos favoráveis e que se tenham adaptado ao estilo de ensino do professor. Também o facto de, em geral, os alunos das amostras estudadas apresentarem, na generalidade, preferências fracas ou moderadas por determinada dimensão dos estilos de aprendizagem pode levar a que, mesmo que os professores utilizem estratégias contrárias ao seu modelo preferido, os alunos não sejam por isso grandemente afectados. Face ao exposto, pensamos poder concluir não existir desajustamento entre os métodos de ensino praticados pelos professores envolvidos nas experiências e os estilos de aprendizagem preferenciais dos alunos.

Também se concluiu que as tarefas de programação habitualmente propostas pelos professores apresentam níveis de dificuldades desajustados ao nível cognitivo do aluno. Se tomarmos com referência a taxonomia dos objectivos educacionais de Bloom, muitos dos alunos apenas conseguem realizar actividades do nível de Conhecimento e Compreensão e as actividades habitualmente propostas pelos professores situam-se, frequentemente, a partir do nível de Aplicação. A maioria dos alunos consegue traduzir pedaços de código isolados mas não os consegue, muitas vezes, interpretar ou generalizar, revelando maiores dificuldades em conseguir juntar as partes de código e interpretá-las de forma interligada. As dificuldades mais evidentes consistem em implementar o núcleo de um programa completo, mesmo que similar a programas fornecidos.

Relativamente ao factor F3, os estudos realizados não nos permitem afirmar que os métodos de estudo utilizados pelos alunos são ou não adequados para a aprendizagem da programação. Porém, constatámos que, em duas amostras de um estudo, as abordagens superficiais e memorísticas estavam correlacionadas com os maus desempenhos a programação. Adicionalmente, os estudos revelaram que certos comportamentos e atitudes face ao estudo, em especial as percepções pessoais de competência e alguns aspectos motivacionais estão fortemente relacionados com os desempenhos a programação.

Relativamente ao factor F4, os estudos evidenciaram que as percepções pessoais dos alunos das amostras estudadas estavam fortemente relacionadas com os seus desempenhos obtidos nas disciplinas introdutórias de programação, bem como com os resultados obtidos às actividades de programação realizadas. Este aspecto forneceu-nos fortes indícios no sentido de afirmar que os alunos com melhores percepções pessoais são os que melhores desempenhos obtêm nas disciplinas de programação.

Apesar de os instrumentos aplicados para diagnosticar o tipo de estudo realizado pelo aluno, não revelar dados preocupantes, continuamos convictos de que a aprendizagem de programação, se distancia da aprendizagem de outras disciplinas pelo que requer uma

abordagem ao estudo muito mais profunda do que aquela que a maioria dos alunos evidencia.

8.1.2 Proposta pedagógica

A realização dos estudos permitiu averiguar certas convicções a fim de edificar uma nova proposta de ensino e aprendizagem de programação. Assim, a grande motivação deste trabalho baseou-se na crença de podermos interferir no processo de ensino da programação com o intuito de melhorar o panorama. Desta forma, cremos que os alunos podem ser ajudados nas suas tarefas de aprendizagem de programação se as condições de ensino e aprendizagem forem diferentes. Para tal, os professores e instituições deveriam ter em consideração alguns aspectos que consideramos relevantes, nomeadamente no que respeita à estrutura lectiva e às características e conhecimentos dos alunos.

Relativamente aos métodos de ensino, apesar de não termos realizado estudos sobre a abordagem utilizada para o desenvolvimento de actividades de programação em sala de aula, temos conhecimento das práticas adoptadas por muitos professores. As nossas convicções e os estudos da literatura levam-nos também a considerar que o tipo de actividades normalmente propostas, consistindo no desenvolvimento de programas completos desde uma fase inicial de aprendizagem de programação, se encontra muito afastado dos níveis de desenvolvimento cognitivo dos alunos. Este desajustamento atinge muitos alunos. Geralmente, no contexto de disciplinas de programação existem poucos alunos de nível médio, sendo a maioria caracterizada por níveis muito baixos e alguns por níveis elevados. Contudo, a generalidade dos professores lança actividades de nível médio, falhando uns e outros, desmotivando a generalidade dos alunos e o próprio professor. A proposta pedagógica apresentada, relativamente a este aspecto, está essencialmente orientada no sentido de propor a utilização de actividades baseadas numa taxonomia de objectivos educacionais. Consideramos que esta aproximação apresenta vantagens pedagógicas, para alunos e professores, pelos motivos já referidos neste documento.

Apesar de a proposta não ter sido testada na sua totalidade, pensamos que inclui um conjunto de aspectos pertinentes para uma nova abordagem de ensino/aprendizagem bem sucedida. Esperamos que nós próprios ou outros possam utilizar este trabalho para confirmar as nossas conclusões e reflexões, traduzindo-se em progressos no ensino e aprendizagem de programação.

8.2 Publicações

Deste trabalho resultou um conjunto de publicações de seguida apresentadas. Destas publicações destacam-se essencialmente dois tipos. O primeiro grupo de publicações dizem essencialmente respeito às dificuldades referentes à problemática estudada, analisadas nos estudos realizados. O segundo grupo de publicações consiste principalmente em propostas de solução para ultrapassar as dificuldades inerentes à educação em programação. Inclui essencialmente propostas de ferramentas desenvolvidas ou em desenvolvimento pelo grupo de investigação em que nos integramos e propostas de desenvolvimentos futuros. Do primeiro grupo referido destacam-se as seguintes publicações:

- Carmo, L., Gomes, A., Pereira, F. & Mendes, A. J. (2006). Learning styles and problem solving strategies**.** *Proceedings of the 3rd E-Learning Conference – Computer Science Education*. Coimbra, Portugal.

  Neste artigo foi publicado parte do estudo A, referente às estratégias de resolução de problemas utilizadas pelos alunos, com determinado perfil de estilos de aprendizagem.

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

  Neste artigo foi publicado parte do estudo A, referente às dificuldades de resolução de problemas e falta de conhecimentos matemáticos elementares evidenciados pelos alunos com dificuldades a programação.

- Gomes, A. & Mendes, A. J. (2007). Problem solving in programming. *Proceedings of the 19th Annual Workshop of the Psychology of Programming Interest Group*. Joensuu, Finland.

  Neste artigo foi publicado o estado da arte realizado sobre resolução de problemas. Esta análise permitiu enumerar um conjunto de importantes princípios para o ensino e aprendizagem de estratégias de resolução de problemas e sua inclusão num sistema computacional. Este artigo suscitou interesse por parte de um editor, a fim de integrar a secção sobre resolução de problemas do livro *Managing Software Professionals,* pelo que teve a seguinte republicação: Gomes, A. & Mendes, A. J. (2009). Problem solving in programming. In G. P. Sudhakar (eds.), *Managing Software Professionals* (pp. 20-38). ICFAI University Press.

- Gomes, A. & Mendes, A. J. (2008). A study on student's characteristics and programming learning. *EDMEDIA'08: Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications 2008* (pp. 2895-2904). AACE. Vienna, Austria.

  Neste artigo foi abordado o estudo B, incluindo os aspectos referentes aos estilos de aprendizagem dos alunos, as suas capacidades em termos de resolução de problemas, as suas competências matemáticas e os seus níveis motivacionais e respectiva correlação com os desempenhos a programação.

- Pacheco, A., Gomes, A., Henriques, J., Almeida, A. & Mendes, A. J. (2008). Mathematics and Programming: Some studies. *CompSysTech'08: Proceedings of the International Conference on Computer Systems and Technologies*. Gabrovo, Bulgaria.

  Neste artigo foram publicados os aspectos referentes às dificuldades de resolução de problemas e falta de conhecimentos matemáticos elementares evidenciados pelos alunos com dificuldades a programação pertencentes ao estudo A, estudo B e estudo C.

- Pacheco, A., Gomes, A., Henriques, J., Almeida, A. & Mendes, A. J. (2008). A study on basic mathematics knowledge for the enhancement of programming learning skills. *IEEIII'2008: Proceedings of the Informatics Education Europe III Conference* (pp. 67-79). Venice, Italy.

  Neste artigo foi publicado o estudo C, referente às dificuldades de resolução de problemas e falta de conhecimentos matemáticos elementares evidenciados pelos alunos com dificuldades a programação, e o recurso à utilização de uma taxonomia para melhor identificar as dificuldades dos alunos.

- Gomes, A. & Mendes, A. J. (2009). Bloom's taxonomy based approach to learn basic programming. *EDMEDIA'09: Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications 2009* (pp. 2547-2554). AACE. Honolulu, Hawaii, USA.

  Neste artigo foi publicado o estudo D, referente às dificuldades de resolução de problemas de programação, e o recurso à utilização de uma taxonomia para melhor identificar as dificuldades dos alunos.

- Gomes, A. & Mendes, A.J. (2010). A study on student performance in first year CS courses. *ITiCSE'10: Proceedings of the 15th Annual Conference on Innovation and Technology in Computer Science Education*. Ankara, Turkey.

  Neste artigo foi avaliada a relação entre os resultados académicos de uma disciplina introdutória de programação e as restantes disciplinas do primeiro ano de uma licenciatura em informática, dando-se particular destaque às disciplinas de base matemática.

- Gomes, A. & Mendes, A. J. (2010). Studies and proposals about initial programming learning. *FIE'2010: Proceedings of the Frontiers In Education 2010*. Washington D.C., USA.

  Neste artigo são apresentados os factores de investigação estudados neste trabalho, referidos os estudos que lhe dão resposta e apresentada a proposta de ensino e aprendizagem de programação sugerida neste trabalho.

Outros estudos realizados encontram-se a aguardar publicação.

Do segundo grupo referido destacam-se as seguintes publicações:

- Marcelino, M. J., Tosheva, I., Dobrodzhaliev, Y., Gomes, A. & Mendes, A. J. (2006). A Web-based approach to support initial algorithmic procedural programming learning. *Proceedings of 3rd E-Learning Conference – Computer Science Education*. Coimbra, Portugal.

  Neste artigo é apresentada uma nova ferramenta o SICAS-W, consistindo num desenvolvimento do SICAS, para suportar a aprendizagem inicial de programação. Esta nova ferramenta, para além de baseada na *web* inclui alguns instrumentos estatísticos para avaliar o desempenho do aluno.

- Gomes, A. & Mendes, A. J. (2007). An environment to improve programming education. *Proceedings of the International Conference on Computer Systems and Technologies*. Rousse, Bulgaria.

  Neste artigo são descritas diversas ferramentas usadas com algum sucesso ao longo do tempo para suportar o ensino-aprendizagem de programação e apresentada uma proposta de um sistema computacional, baseado no nível cognitivo do aluno e no seu perfil preferencial de aprendizagem.

- Gomes, A. & Mendes, A. J. (2007). Learning to program – difficulties and solutions. *Proceedings of the International Conference on Engineering Education*. Coimbra, Portugal.

  Neste artigo, são apresentadas as dificuldades inerentes ao ensino/aprendizagem de programação e sugeridas estratégias para colmatá-las com base na proposta de um ambiente computacional centrado essencialmente em actividades de resolução de problemas.

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

  Neste artigo são apresentadas as dificuldades inerentes ao ensino/aprendizagem de programação e sugeridos tipos de actividades que uma ferramenta de e-learning deveria suportar de forma a possibilitar o ensino-aprendizagem de programação abrangendo todos os estilos de aprendizagem.

- Areias, C., Gomes**, A.** & Mendes, A. J. (2007). Learning to program with ProGuide. *Proceedings of the International Conference on Engineering Education*. Coimbra, Portugal.

  Neste artigo são referidos alguns problemas associados à aprendizagem inicial de programação com que os alunos mais fracos se deparam e é sugerida uma ferramenta, que integra o SICAS, para apoiá-los.

- Gomes, A., Areias, C., Henriques, J. & Mendes, A. J. (2008). Aprendizagem de programação de computadores: dificuldades e ferramentas de suporte. *Revista Portuguesa de Pedagogia*, *42* (2), 161-179 (artigo convidado).

  Neste artigo, são apresentadas as dificuldades inerentes ao ensino/aprendizagem de programação e sugeridas ferramentas para as minorar, destacando-se particularmente as vantagens do SICAS desenvolvido pela investigadora e outras ferramentas desenvolvidas por este grupo de investigação.

- Gomes, A., Henriques, J. & Mendes, A. J. (2008). Uma proposta para ajudar alunos com dificuldades na aprendizagem inicial de programação de computadores. *Educação, Formação & Tecnologias*, *1*, 93-103 (artigo convidado).

  Neste artigo, são apresentadas as dificuldades inerentes ao ensino/aprendizagem de programação e sugerida uma estratégia *top-down* para ensinar programação através do ensino da implementação de um jogo.

- Santos, A., Gomes, A. & Mendes, A. J. (2008). E-Learning platform adapted to programming teaching/learning. *EDMEDIA'08: Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications 2008*. AACE. Vienna, Austria.

  Neste artigo são apresentadas as dificuldades inerentes ao ensino/aprendizagem de programação e sugerida uma ferramenta de *e-learning* que recorre a uma nova versão do SICAS. É salientada a organização e o conjunto de características e funcionalidades desejáveis desta ferramenta.

- Santos, A., Gomes, A. & Mendes, A. J. (2010). Integrating New Technologies and Existing Tools to Promote Programming Learning. *Algorithms*, *3* (2), 183-196 (artigo convidado para o número especial sobre *Innovative Software Tools for Learning and Teaching Computer Programming Concepts*).

  Neste artigo, são apresentadas as ferramentas desenvolvidas pelo nosso grupo de investigação para apoiar o ensino/aprendizagem de programação destacando particularmente uma nova ferramenta em desenvolvimento, tendo como base o SICAS, considerando determinados aspectos entretanto investigados.

8.3 Áreas de desenvolvimento potencial

Os resultados da aplicação desta proposta poderão ditar muitos percursos futuros a explorar. A área de ensino e aprendizagem de programação inclui, actualmente, uma enorme diversidade de tópicos de pesquisa. Assim, acreditamos na utilidade da continuação deste trabalho em diferentes linhas de investigação. Relativamente às dez áreas de investigação referentes a este assunto, mencionadas neste documento, consideramos particularmente útil as respeitantes à "Compreensão do aluno", especialmente nas subáreas de "Estudos psicológicos", "Misconceptions" e "Estudos fenomenográficos". Também cremos na importância de desenvolvimentos na área de "Métodos de ensino", com particular relevância nas subáreas de "Currículo invertido" e "Padrões". Pensamos que os estudos nas áreas anteriormente mencionados podem trazer um conjunto de mais-valias para futuros desenvolvimentos nas não menos importantes áreas de "Animação, visualização e simulação" bem como nas áreas de "Tecnologia educacional e inclusão de desenvolvimentos e tecnologias".

Relativamente à área de "Compreensão do aluno", consideramos elemento de primordial importância a definição de um conjunto de funções cognitivas ou competências

que servissem de base a uma boa capacidade de resolução de problemas de programação. Tendo definidas essas competências, seria também importante averiguar as suas dependências a fim de verificar se seriam necessárias todas ao mesmo tempo ou em etapas distintas ou quais é que se deviam treinar primeiro. O estabelecimento de uma hierarquia de competências permitiria definir programas para diagnosticar e posteriormente treinar essas funções. Na área da Psicologia facilmente se encontram definidos modelos/teorias cognitivas referentes ou implicados na resolução de problemas. Duas das áreas que têm fornecido valiosos contributos para o entendimento dos processos cognitivos são a Psicologia Cognitiva (processamento da informação) e a Psicologia do Desenvolvimento (desenvolvimento da inteligência). Desta forma consideramos que uma possível e útil linha de investigação possa ser direccionada no sentido de pesquisar, analisar e/ou desenvolver instrumentos de avaliação cognitiva credíveis, de forma a detectar e posteriormente treinar as competências pretendidas.

Outro aspecto que consideramos de grande utilidade investigar, referente à área de "Compreensão do aluno" prende-se com as dificuldades que os alunos apresentam relativamente à compreensão dos conceitos ensinados em programação. Relativamente a este factor podem ser seguidos diferentes percursos de investigação. Uma possibilidade consiste na utilização da fenomenografia (Marton e Booth, 1997), uma abordagem particular de investigação para estudar problemas de variadas áreas. Consiste num método empírico que segue uma abordagem qualitativa, tendo as suas raízes na investigação pedagógica. Num projecto de investigação fenomenográfica, o investigador analisa e descreve os diferentes modos pelos quais um fenómeno (ou um conjunto de fenómenos) é compreendido, experienciado, percebido ou aprendido, do ponto de vista do aprendiz. Uma vez que um estudo fenomenográfico toma em consideração a perspectiva dos alunos, dá "voz" aos alunos. Sendo assim, não só o que o aluno aprende mas também a forma como aprende e os objectivos que tem podem ser explorados. Achamos esta perspectiva bastante interessante, pois ensinar bem implica aprender sobre a forma como os alunos aprendem. A abordagem fenomenográfica tem por objectivo interpretar, descrever e categorizar como é que um fenómeno é interpretado por um grupo de aprendizes. Cada uma das categorias resultantes permite descrever uma certa forma pela qual o fenómeno sob investigação é compreendido. Todas juntas, as categorias, descrevem uma variedade de entendimentos que podem ser encontradas num grupo. Por vezes, estabelecem-se paralelismos entre a fenomenografia e as teorias da aprendizagem baseadas na psicologia. Porém, as diferenças são importantes. Enquanto que a psicologia discute o que é aprendido, independentemente do assunto que é aprendido e do contexto do que é aprendido, a fenomenografia estuda a experiência de aprender algo, num cenário e contexto particular. Actualmente discute-se mais a combinação de duas teorias, a fenomenografia (perceber como é que os conceitos chave são entendidos

pelos alunos, feito através de entrevistas semi-estruturadas) e a *Variation Theory* (perceber que aspectos de determinado problema precisam de ser variados de forma a proporcionar um completo entendimento aos alunos).

Outra possibilidade, consiste num estudo mais aprofundado sobre as *misconceptions* que os alunos apresentam relativamente ao entendimento dos diversos conceitos de programação. Apesar de se encontrarem na literatura estudos genéricos sobre o assunto, pensamos que esta é uma área que carece de investigações mais detalhadas. Relacionado com este aspecto consideramos ainda importante a investigação relativa aos erros semânticos e lógicos cometidos pelos alunos enquanto programam.

Pensamos que estas duas últimas orientações de investigação mencionadas poderão constituir abordagens interessantes, no sentido de permitir obter indicações úteis para a elaboração de uma ferramenta poderosa para melhorar o processo de ensino/aprendizagem. Em especial consideramos útil a existência de um conjunto de visualizações que permitam concretizar as abstracções envolvidas nos diversos conceitos de programação. Este é um aspecto que poderá merecer particular atenção uma vez que o efeito educativo de observar visualizações é muitas vezes menor do que o esperado, pelo que outro percurso de investigação poderá ser direccionado para a área de "Animação, simulação e visualização" incidindo nos aspectos e abordagens que tornam essas representações eficazes. Relativamente a este assunto, consideramos ainda de extrema importância a pesquisa, estudo e desenvolvimento de metáforas significativas conducentes ao entendimento dos diversos assuntos.

Compreender a mente humana é um dos desafios mais antigos das ciências cognitivas. Pensamos que a exploração desta área poderá constituir outro rumo de investigação que pode trazer valiosos contributos nomeadamente no que concerne às teorias sobre a aquisição de conhecimento, possibilitando a exploração de modelos cognitivos para a compreensão de programas. Em especial, será útil perceber as representações mentais que utilizamos internamente quando tentamos compreender um programa ou algoritmo e de acordo com os nossos conhecimentos como é que conseguimos adquirir novo conhecimento, de forma a compreender algoritmos mais elaborados. Desta forma, consideramos de utilidade a pesquisa sobre métodos e suportes cognitivos existentes e/ou desejáveis para auxiliar o processo de entendimento de programas. Este aspecto poderá ser particularmente útil para ajudar a definir novas abordagens de ensino e aprendizagem.

Relativamente à área de "Métodos de ensino" e, de certa forma, relacionado com o aspecto anteriormente referido consideramos também de grande utilidade a existência de esquemas genéricos de código que, representem cenários típicos de programação. Pelo que,

um outro percurso de investigação poderá consistir na investigação de planos ou padrões pedagógicos de programação. Estes padrões basicamente consistiriam em pequenos *templates*, projectados por peritos em programação que descreveriam soluções de programação para problemas que ocorrem frequentemente no desenvolvimento de programas. Outra característica destes *templates* seria a de incorporarem regras gerais de estilo de programação eficazes e elegantes, de forma a constituírem soluções estáveis, eficientes, portáveis, reutilizáveis e de fácil utilização. Os padrões poderiam assim ajudar o aprendiz de programação na aprendizagem de estratégias gerais ou abstracções, das mais básicas às mais avançadas, bem como facilitar a interiorização da sintaxe de determinada linguagem. A base genérica desta ideia consiste em, ao resolver um problema de programação, o aluno ter de escolher um padrão ou solução geral de programação, construindo, a partir dele, a solução do problema. Consideramos adicionalmente benéfica a existência de um ambiente que permita ao aluno testar os seus exercícios com base nos padrões fornecidos.

Relativamente à área genérica de "Métodos de ensino" e particularmente referente às estratégias de "Compreensão de programas", visionamos essencialmente dois tipos de abordagens, estratégias *Top-Down* (começam pelos conceitos mais abstractos e gerais de um programa e, através de um processo de análise, decomposição ou refinamentos sucessivos chegam ao detalhe) ou estratégias *Bottom-Up* (começam pelos detalhes de um programa, chegando aos níveis mais altos de abstracção, por agregações ou sínteses sucessivas). Neste domínio acreditamos nas vantagens educativas de uma estratégia *Top-Down*, relacionada também com a área do "Currículo invertido". Sobre este aspecto consideramos que uma boa linha de investigação poderá passar pelo ensino da programação através de um sistema que através de grande interactividade demonstre a implementação de um jogo computacional, na perspectiva do ensino/aprendizagem de programação. O intuito seria o de mostrar aos alunos para que serve e como se faz um programa, neste caso um jogo, devido às vantagens lúdicas e motivacionais associadas. Esse sistema seria coadjuvado por três bonecos animados com funções distintas, um para ilustrar o jogo do ponto de vista do utilizador/jogador, outro para ilustrar a forma lógica (programação) de representação do jogo e outro ainda para ilustrar a forma física (armazenamento interno) de representação do jogo. Esse ambiente multimédia seria baseado na resolução de problemas e algoritmos colocando em segundo plano os detalhes sintácticos da linguagem. Consistiria na incorporação de vários tipos de actividades de resolução de problemas (numa 1ª fase com um carácter mais lúdico e de diversos domínios), de forma a atrair e estimular os alunos, progredindo gradualmente para domínios mais específicos e próximos da programação. O objectivo final consistiria na construção de algoritmos, pretendendo transformar formalizações desenvolvidas em fases anteriores em procedimentos sistemáticos. Os problemas apresentados aos alunos passariam

a exigir soluções mais elaboradas, nas quais cada vez mais estaria inerente o acto de explicitar procedimentos. A última fase culminaria com a resolução de algoritmos de uma forma estruturada utilizando para isso o SICAS (com melhoramentos) ou um ambiente similar. O aspecto principal residiria em, através de um ambiente estimulante e atractivo, averiguar o progresso do aluno, nos aspectos de capacidade de abstracção, de raciocínio lógico, da solução de problemas e da autonomia cognitiva. Cada uma das fases apontadas seriam sempre aplicadas de acordo com o estado actual de conhecimento do aluno e do seu perfil preferencial de aprendizagem.

Outra abordagem que consideramos de grande interesse passaria pela exploração do comportamento dos alunos perante "Métodos de ensino" alternativos, utilizando por exemplo robôs.

Cap. 9

Referências

Ackerman, J. M. (1993). The promise to learn. *Written Comunication, 10* (3), 334-370.

ACM. (2001). *Computing Curricula 2001: Computer Science (Report from The Joint Task Force on Computing Curricula).* Obtido em 1 de Abril de 2010, de Curricula Recommendations - Association for Computing Machinery: http://www.acm.org/education/education/education/curric_vols/cc2001.pdf

ACM. (2005). *Computing Curricula 2005: The Overview Report (Report from The Joint Task Force for Computing Curricula 2005).* Obtido em 1 de Abril de 2010, de Curricula Recommendations - Association for Computing Machinery: http://www.acm.org/education/education/curric_vols/CC2005-March06Final.pdf

ACM. (2008). *Computer Science Curriculum 2008: An Interim Revision of CS 2001 (Report from the Interim Review Task Force)*. Obtido em 01 de Abril de 2010, de Curricula Recommendations - Association for Computing Machinery: http://www.acm.org//education/curricula/ComputerScience2008.pdf

Aiken, A. (1994). *MOSS - A System for Detecting Software Plagiarism*. Obtido em 1 de Abril de 2010, de http://theory.stanford.edu/~aiken/moss/

Ala-Mutka, K. M. (2005). A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education, 15* (2), 83-102.

Ala-Mutka, K. M., & Järvinen, H.-M. (2004). Assessment Process for Programming Assignments. *ICALT'04: Proceedings of the Fourth IEEE International Conference on Advanced Learning Technologies* (pp. 181-185). IEEE Computer Society. Joensuu, Finland.

Allen, E. L., & Mourtos, N. J. (2000). Using learning styles preferences data to inform classroom teaching and assessment activities. *FIE'2000: Proceedings of the 30th Annual Frontiers in Education. 2*, pp. S2B-6. IEEE Computer Society. Kansas City, MO, USA.

Allen, E., Cartwright, R., & Stoler, B. (2002). DrJava: a lightweight pedagogic environment for Java. *SIGCSE Bulletin, 34* (1), 137-141.

Allert, J. (2004). Learning Style and Factors Contributing to Success in an Introductory Computer Science Course. *ICALT'04: Proceedings of the Fourth IEEE International Conference on Advanced Learning Technologies* (pp. 385-389). IEEE Computer Society. Joensuu, Finland.

Allwood, C. M. (1986). Novices on the computer: a review of the literature. *International Journal of Man-Machine Studies, 25*, 633-658.

Almeida, A. C. (2004). *Cognição como Resolução de Problemas: Novos horizontes para a investigação e intervenção em Psicologia e Educação.* Tese de Doutoramento, Faculdade de Psicologia e Ciências da Educação, Universidade de Coimbra, Coimbra, Portugal.

Alphonce, C. G. (2003). *"Killer Examples" for Design Patterns and Objects First*. Obtido em 01 de Abril de 2010, de "Killer Examples" workshops: http://www.cse.buffalo.edu/~alphonce/KillerExamples/OOPSLA2002/

Alves, V., Cruz, V., & Fonseca, V. (2002). *Educação Cognitiva e Aprendizagem.* Porto Editora. Porto, Portugal.

Anderson, J. R. (1983). *The architecture of cognition.* Harvard University Press. Cambridge, MA, USA.

Anderson, J. R. (1985). *Cognitive Psychology and its Implications* (2nd ed.). Freeman. New York, USA.

Anderson, J. R., & Reiser, B. J. (1985). The LISP tutor: it approaches the effectiveness of a human tutor. *Byte, 10* (4), pp. 159-175.

Anderson, L., Krathwohl, D., Airasian, P., Cruikshank, K., Mayer, R., Pintrich, P., et al. (2001). *A Taxonomy for Learning and Teaching and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives.* Addison Wesley Longman, Inc.. New York, USA.

Anderson, R., Anderson, R., Simon, B., Wolfman, S. A., VanDeGrift, T., & Yasuhara, K. (2004). Experiences with a tablet PC based lecture presentation system in computer science courses. *SIGCSE Bulletin, 36* (1), 56-60.

Anjaneyulu, K. S. (1994). Bug analysis of Pascal programs. *SIGPLAN Notices, 29* (4), 15-22.

Arnow, D., & Barshay, O. (1999). WebToTeach: an interactive focused programming exercise system. *FIE'99: Proceedings of the 29th Frontiers in Education Conference. 1*, pp. 12A9/39-12A9/44. IEEE Computer Society. San Juan, Puerto Rico.

Astrachan, O., & Reed, D. (1995). AAA and CS 1: the applied apprenticeship approach to CS 1. *SIGCSE Bulletin, 27* (1), 1-5.

Astrachan, O., Mitchener, G., Berry, G., & Cox, L. (1998). Design patterns: an essential component of CS curricula. *SIGCSE Bulletin, 30* (1), 153-160.

Astrachan, O., Smith, R., & Wilkes, J. (1997). Application-based modules using apprentice learning for CS 2. *SIGCSE Bulletin, 29* (1), 233-237.

Atchison, W. F., Conte, S. D., Hamblen, J. W., Hull, T. E., Keenan, T. A., Kehl, W. B., et al. (1968). Curriculum'68: Recommendations for academic programs in computer science: a report of the ACM curriculum committee on computer science. *Communications of the ACM, 11* (3), 151-197.

Austing, R. H., Barnes, B. H., Bonnette, D. T., Engel, G. L., & Stokes, G. (1979). Curriculum'78: recommendations for the undergraduate program in computer science - a report of the ACM curriculum committee on computer science. *Communications of the ACM, 22* (3), 147-166.

Ayre, M., & Nafalski, A. (2000). Recognising diverse learning styles in teaching and assessment of electronic engineering. *FIE '00: Proceedings of the 30th Annual Frontiers in Education. 1*, pp. T2B/18-T2B/23. IEEE Computer Society. Kansas City, MO, USA.

Azuma, M., Coallier, F., & Garbajosa, J. (2003). How to Apply the Bloom Taxonomy to Software Engineering. *STEP '03: Proceedings of the Eleventh Annual International Workshop on Software Technology and Engineering Practice* (pp. 117-122). IEEE Computer Society. Amsterdam, The Netherlands.

Baecker, R. (1968). Experiments in On-Line Graphical Debugging: The Interrogation of Complex Data Structures. *Proceedings of First Hawaii International Conference on the System Sciences*, (pp. 128-129). Hawaii, USA.

Baecker, R. (1975). Two systems which produce animated representations of the execution of computer programs. *SIGCSE Bulletin, 7* (1), 158-167.

Baecker, R. (1981). Sorting Out Sorting. *30 minute colour sound film presented at ACM SIGGRAPH `81 and excerpted in ACM SIGGRAPH Video Review #7, 1983* . Morgan Kaufmann, Publishers. Los Altos, CA, USA.

Baecker, R. (1998). Sorting Out Sorting: A Case Study of Software Visualization for Teaching Computer Science. In J. Stasko, J. Domingue, M. H. Brown, & B. A. Price (Edits.), *Software Visualization: Programming as a Multimedia Experience* (pp. 369-381). MIT Press. Cambridge, MA, USA.

Baecker, R., & Marcus, A. (1986). Design principles for the enhanced presentation of computer program source text. *SIGCHI Bulletin, 17* (4), 51-58.

Bagert, D. J., Calloni, B. A., & Haiduk, H. (1996). Iconic vs. text-based programming in the introductory programming sequence. *Proceedings of the ASEE Annual Conference (CDROM).* Washington DC, USA.

Baleche, F. L. (2003). *Estilos de aprendizagem: um caminho para o educador na prática pedagógica.* Dissertação de mestrado, Universidade Federal de Santa Catarina, Brasil.

Ball, S. (1977). *Motivation in Education.* Academic Press. New York, USA.

Bandler, R., & Grinder, J. (1979). *Frogs into Princes: Neuro Linguistic Programming.* Real People Press. Moab, Utah, USA.

Bandura, A. (Ed.). (1995). *Self-efficacy in changing societies.* Cambridge University Press. Cambridge, UK.

Barbe, W. B., & Milone, M. N. (1981). What We Know about Modality Strengths. *Educational Leadership, 38* (5), 378-380.

Barbe, W. B., Swassing, R. H., & Milone, M. N. (1979). *Teaching Through Modality Strengths: Concepts and Practices.* Zaner Bloser. Columbus, Ohio, USA.

Barnes, D. J., & Kölling, M. (2006). *Objects First With Java: A Practical Introduction Using BlueJ* (3rd Edition ed.). Prentice-Hall, Inc.. Upper Saddle River, NJ, USA.

Barros, L. N., Mota, A. P., Delgado, K. V., & Matsumoto, P. M. (2005). A tool for programming learning with pedagogical patterns. *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange*, (pp. 125-129). San Diego, CA, USA.

Bateson, A., Alexander, R. A., & Murphy, M. D. (1987). Cognitive processing differences between novice and expert computer programmers. *International Journal of Man-Machine Studies, 26* (6), 649-660.

Bayliss, J. D., & Strout, S. (2006). Games as a "flavor" of CS1. *SIGCSE Bulletin, 38* (1), 500-504.

Beaubouef, T., & Mason, J. (2005). Why the high attrition rate for computer science students: some thoughts and observations. *SIGCSE Bulletin, 37* (2), 103-106.

Beck, K. (2000). *Extreme programming explained: embrace change.* Addison-Wesley. Boston, MA, USA.

Beck, K. (2003). *Test-Driven Development: By Example.* Addison-Wesley. Boston, MA, USA.

Becker, K. (2006). How much choice is too much? *SIGCSE Bulletin, 38* (4), 78-82.

Ben-Ari, M. (2001). Constructivism in Computer Science Education. *Journal of Computers in Mathematics and Science Teaching, 20* (1), 45-73.

Benford, S., Burke, E., Foxley, E., Gutteridge, N., & Zin, A. M. (1993). Ceilidh: A course administration and marking system. *CBLIS'93: Proceedings of the International Conference on Computer-Based Learning in Science*, (pp. 364-372). Vienna, Austria.

Bennedsen, J. (2008). *Teaching and Learning Introductory Programming – A Model-Based Approach.* PhD Thesis, University of Oslo, Norway.

Bennedsen, J., & Caspersen, M. E. (2003). Rationale for the Design of a Web-based Programming Course for Adults. *ICOOL 2003: Proceedings of the International Conference on Open and Online Learning.* University of Mauritius, Reduit, Mauritius.

Bennedsen, J., & Caspersen, M. E. (2005a). An investigation of potential success factors for an introductory model-driven programming course. *ICER'05: Proceedings of the First International Workshop on Computing Education Research* (pp. 155-163). ACM. New York, NY, USA.

Bennedsen, J., & Caspersen, M. E. (2005b). Revealing the programming process. *SIGCSE Bulletin, 37* (1), 186-190.

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

Bennedsen, J., & Eriksen, O. (2006). Categorizing Pedagogical Patterns by Teaching Activities and Pedagogical Values. *Computer Science Education, 16* (2), 157-172.

Bennedsen, J., & Schulte, C. (2007). What does 'Objects-First' mean? An International Study of Teachers' Perceptions of Objects-First. In R. Lister, & Simon (Ed.), *Koli Calling 2007: Proceedings of the Seventh Baltic Sea Conference on Computing Education Research. 88*, pp. 21-29. ACS. Koli National Park, Finland.

Bentley, J., & Kernigham, B. (1991). A System for Alghorithm Animation: Tutorial and User Manual. *Computing Systems, 4* (1), 5-30.

Bergin, J. (2007). *Beyond Karel J Robot: A Gentle Introduction to the Art of Object-Oriented Programming in Java* (Vol. 2). Dream Songs Press. Redwood City, CA, USA.

Bergin, J., Brodlie, K., Patiño-Martínez, M., McNally, M., Naps, T., Rodger, S., et al. (1996). An overview of visualization: its use and design (Report of the Working Group in Visualization). *SIGCSE Bulletin, 28* (SI), 192-200.

Bergin, J., Caristi, J., Dubinsky, Y., Hazzan, O., & Williams, L. (2004). Teaching software development methods: the case of extreme programming. *SIGCSE Bulletin, 36* (1), 448-449.

Bergin, J., Stehlik, M., Roberts, J., & Pattis, R. (1997). *Karel++: A Gentle Introduction to the Art of Object-Oriented Programming.* John Wiley & Sons, Inc.. New York, USA.

Bergin, J., Stehlik, M., Roberts, J., & Pattis, R. (2005). *Karel J Robot: A Gentle Introduction to the Art of Object-Oriented Programming in Java.* Dream Songs Press. Redwood City, CA, USA.

Bergin, S., Reilly, R., & Traynor, D. (2005). Examining the role of self-regulated learning on introductory programming performance. *ICER'05: Proceedings of the First International Workshop on Computing Education Research* (pp. 81-86). ACM. Seattle, WA, USA.

Bettini, L., Crescenzi, P., Innocenti, G., Loreti, M., & Cecchi, L. (2004). An Environment for Self-Assessing Java Programming Skills in Undergraduate First Programming Courses. *ICALT'04: Proceedings of the Fourth IEEE International Conference on Advanced Learning Technologies* (pp. 161-165). IEEE Computer Society. Joensuu, Finland.

Bielikova, M. (2006). An adaptive web-based system for learning programming. *International Journal of Continuing Engineering Education and Life-Long Learning, 16* (1/2), 122-136.

Biggs, J. B. (1993). What do inventories of students' learning processes really measure? A theoretical review and clarification. *British Journal of Educational Psychology, 63* (1), 3-19.

Biggs, J. B. (2003). *Teaching for Quality Learning at University: What the student does* (2nd ed.). Society for Research into Higher Education & Open University Press. Philadelphia, PA, USA.

Biggs, J. B., & Collis, K. F. (1982). *Evaluating the quality of learning : the SOLO taxonomy (structure of the observed learning outcome).* Academic Press. New York, USA.

Blank, D., Meeden, L., & Kumar, D. (2003). Python robotics: an environment for exploring robotics beyond LEGOs. *SIGCSE Bulletin, 35* (1), 317-321.

Bloom, B. S. (1984). *Bloom's Taxonomy*. Obtido em 01 de Abril de 2010, de University of Victoria - Learning Skills: http://www.coun.uvic.ca/learning/exams/blooms-taxonomy.html

Bloom, B. S., Engelhart, M. D., Furst, E. J., Hill, W. H., & Krathwohl, D. R. (1956). *Taxonomy of Educational Objectives, Handbook I: Cognitive Domain.* Longmans, Green and Co Ltd. London, UK.

Bonar, J. (1982). Natural Problem Solving Strategies and Programming Language Constructs. *Proceedings of the Fourth Annual Conference of the Cognitive Science Society.* Ann Arbor, Michigan, USA.

Bonar, J., Cunningham, R., Beatty, P., & Weil, W. (1988). *Bridge: Intelligent Tutoring System with Intermediate Representations.* Technical Report, Learning Research & Development Center, University of Pittsburgh, Pittsburgh, PA, EUA.

Booth, S. (1997). On Phenomenography, Learning and Teaching. *Higher Education Research & Development, 16* (2), 135-158.

Box, I. (2003). Assessing the assessment: an empirical study of an information systems development subject. *ACE'03: Proceedings of the fifth Australasian Conference on Computing Education. 20*, pp. 149-158. Australian Computer Society, Inc.. Adelaide, Australia.

Bransford, J. D., & Stein, B. S. (1984). *The IDEAL problem solver: A guide for improving thinking, learning and criativity.* Freeman. New York, USA.

Bransford, J. D., Brown, A. L., & Cocking, R. R. (Eds.). (2000). *How people learn: Brain, mind, experience, and school.* National Academy Press. Washington, D. C., USA.

Brown, M. H. (1988). Exploring Algorithms Using Balsa-II. *IEEE Computer, 21* (5), 14-36.

Brown, M. H. (1991). Zeus: A system for algorithm animation and multi-view editing. *Proceedings of the 1991 IEEE Workshop on Visual Languages*, (pp. 4-9). Kobe, Japan.

Brown, M. H., & Sedgewick, R. (1985). Techniques for Algorithm Animation. *IEEE Software, 2* (1), 28-39.

Brown, R. W. (2001). Multi-choice versus descriptive examinations. *FIE'01: Proceedings of the 31st Annual Frontiers in Education Conference. 1*, pp. T3A/13-T3A/18. IEEE Computer Society. Reno, NV, USA.

Bruce, C. S., & McMahon, C. A. (2002). *Contemporary Developments in Teaching and Learning Introductory Programming: Towards a Research Proposal.* Teaching and Learning Report, Faculty of Information Technology, Queensland University of Technology, Brisbane, Australia.

Bruce, C. S., Buckingham, L. I., Hynd, J. R., McMahon, C. A., Roggenkamp, M. G., & Stoodley, I. D. (2004). Ways of experiencing the act of learning to program: A phenomenographic study of introductory programming students at university. *Journal of Information Technology Education, 3*, 143-160.

Brusilovsky, P. (1991). Turingal - the language for teaching the principles of programming. *Proceedings of Third European Logo Conference*, (pp. 423-432). Parma, Italy.

Brusilovsky, P. (1993). Program visualization as a debugging tool for novices. *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems* (pp. 29-30). ACM. Amsterdam, The Netherlands.

Brusilovsky, P., & Higgins, C. (2005). Preface to the special issue on automated assessment of programming assignments. *Journal on Educational Resources in Computing (JERIC), 5* (3), Article no. 1.

Brusilovsky, P., & Pathak, S. (2002). Assessing Student Programming Knowledge with Web-based Dynamic Parameterized Quizzes. In P. Barker, & S. Rebelsky (Ed.), *ED-MEDIA'2002: Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications* (pp. 1548-1553). AACE. Denver, Colorado, USA.

Brusilovsky, P., & Sosnovsky, S. (2005). Individualized exercises for self-assessment of programming knowledge: An evaluation of QuizPACK. *Journal on Educational Resources in Computing (JERIC), 5* (3), Article no. 6.

Buck, D., & Stucki, D. J. (2000). Design early considered harmful: graduated exposure to complexity and structure based on levels of cognitive development. *SIGCSE Bulletin, 32* (1), 75-79.

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

Buckley, J., & Exton, C. (2003). Blooms' Taxonomy: A Framework for Assessing Programmers' Knowledge of Software Systems. *IWPC'03: Proceedings of the 11th IEEE International Workshop on Program Comprehension* (p. 165). IEEE Computer Society. Portland, Oregon, USA.

Burgess, G. A. (2005). Introduction to programming: blooming in America. *Journal of Computing Sciences in Colleges, 21* (1), 19-28.

Butcher, D. F., & Muth, W. A. (1985). Predicting performance in an introductory computer science course. *Communications of the ACM, 28* (3), 263-268.

Buxeda, R., Jimenez, L., & Morell, L. (2001). Transforming an Engineering Course to Enhance Student Learning. *Proceedings of the International Conference on Engineering Education.* Oslo, Norway.

Byrne, P., & Lyons, G. (2001). The effect of student attributes on success in programming. *SIGCSE Bullettin, 33* (3), 49-52.

Calabrese, E. (1989). Marta – The "intelligent turtle". In G. Schuyten, & M. Valcke (Ed.), *Proceedings of the Second European Logo Conference*, (pp. 111-127). Ghent, Belgium.

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

Carlisle, M. C., Wilson, T. A., Humphries, J. W., & Hadfield, S. M. (2005). RAPTOR: a visual programming environment for teaching algorithmic problem solving. *SIGCSE Bullettin, 37* (1), 176-180.

Carter, J., & Jenkins, T. (1999). Gender and programming: what's going on? *SIGCSE Bulletin, 31* (3), 1-4.

Caspersen, M. E. (2007). *Educating novices in the skills of programming.* PhD Thesis, Department of Computer Science, University of Aarhus, Denmark.

Caspersen, M. E., & Kölling, M. (2006). A novice's process of object-oriented programming. *OOPSLA'06: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications* (pp. 892-900). ACM. Portland, Oregon, USA.

Caspersen, M. E., Larsen, K. D., & Bennedsen, J. (2007). Mental models and programming aptitude. *SIGCSE Bulletin, 39* (3), 206-210.

Cegielski, C. G., & Hall, D. J. (2006). What makes a good programmer? *Communications of the ACM, 49* (10), 73-75.

Cerqueira, T. C. (2000). *Estilos de Aprendizagem em Universitários.* Tese de Doutoramento, Faculdade de Educação, Universidade Estadual de Campinas, Campinas, SP, Brasil.

Chamillard, A. T. (2006). Using student performance predictions in a computer science curriculum. *SIGCSE Bulletin, 38* (3), 260-264.

Chamillard, A. T., & Karolick, D. (1999). Using learning style data in an introductory computer science course. *SIGCSE Bulletin, 31* (1), 291-295.

Chandhok, R. P., & Miller, P. L. (1989). The design and implementation of the Pascal GENIE. *CSC'89: Proceedings of the 17th Conference on ACM Annual Computer Science Conference* (pp. 374-379). ACM. Louisville, Kentucky, USA.

Chandhok, R. P., Garlan, D. H., Meter, G., Miller, P. L., & Pane, J. (1991). *Pascal Genie (version 1.0).* Chariot Software Group. San Diego, CA, USA.

Chang, S.-K. (Ed.). (1990). *Principles of visual programming systems.* Prentice-Hall, Inc.. Upper Saddle River, NJ, USA.

Chang, T., & Chang, D. (2000). The Role of Myers-Briggs Type Indicator in Electrical Engineering Education. *ICEE'2000: Proceedings of the 2000 International Conference on Engineering Education*, (pp. 1-5). Taipei & Tainan, Taiwan.

Chatley, R., & Timbul, T. (2005). KenyaEclipse: learning to program in eclipse. *SIGSOFT Software Engineering Notes, 30* (5), 245-248.

Cheang, B., Kurnia, A., Lim, A., & Oon, W.-C. (2003). On automated grading of programming assignments in an academic institution. *Computers & Education, 41* (2), 121-131.

Cheung, R. (2006). A web-based learning environment for object-oriented programming. *International Journal of Information and Operations Management Education, 1* (2), 140-157.

Chi, M. T., Feltovich, P. J., & Glaser, R. (1981). Categorization and representations of physics problems by experts and novices. *Cognitive Science, 5*, 121-152.

Chi, M. T., Feltovich, P. J., & Glaser, R. (1981). Expertise in problem solving. In R. Sternberg (Ed.), *Advances in the psychology of human intelligence.* Lawrence Erlbaum Associates. Hillsdale, NJ, USA.

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

Christensen, H. B. (2004). Frameworks: putting design patterns into perspective. *SIGCSE Bulletin, 36* (3), 142-145.

Clancy, M. J. (2004). Misconceptions and Attitudes that Interfere with Learning to Program. In M. Petre, & S. Fincher (Eds.), *Computer Science Education Research* (pp. 85-100). Taylor & Francis Group. London, UK.

Clancy, M. J., & Linn, M. C. (1999). Patterns and pedagogy. *SIGCSE Bulletin, 31* (1), 37-42.

Cockburn, A. (2002). *Agile software development.* Addison-Wesley Longman Publishing Co., Inc.. Boston, MA, USA.

Coffield, F., Moseley, D., Hall, E., & Ecclestone, K. (2004). *Learning styles and pedagogy in post-16 learning: A systematic and critical review.* Learning and Skills Research Centre, Cromwell Press Ltd. Trowbridge, Wiltshire, UK.

Comer, D. E., Gries, D., Mulder, M. C., Tucker, A., Turner, A. J., & Young, P. R. (1989). Computing as a discipline. *Communications of the ACM, 32* (1), 9-23.

Computing Accreditation Commission. (1985). *Criteria for Accrediting Computing Programs: Effective for Evaluations During the 2006-2007 Accreditation Cycle.* ABET Inc. Baltimore, MD, USA.

Conrow, K., & Smith, R. G. (1970). NEATER2: a PL/I source statement reformatter. *Communications of the ACM, 13* (11), 669-675.

Constant, K. P. (1997). Using Multimedia Techniques to Address Diverse Learning Styles in Materials Education. *Journal of Materials Education, 19* (1&2), 1-8.

Coop, R. H., & Brown, L. D. (1970). Effects of cognitive style and teaching method on categories of achievement. *Journal of Educational Psychology, 61* (5), 400-405.

Cooper, S., Cassel, L., Moskal, B., & Cunningham, S. (2005). Outcomes-based computer science education. *SIGCSE Bulletin, 37* (1), 260-261.

Cooper, S., Dann, W., & Pausch, R. (2000). Alice: a 3-D Tool for Introductory Programming Concepts. *Journal of Computing Sciences in Colleges, 15* (5), 107-116.

Cooper, S., Dann, W., & Pausch, R. (2003). Teaching objects-first in introductory computer science. *SIGCSE Bulletin, 35* (1), 191-195.

Costa, S. S., & Moreira, M. A. (1996). Resolução de problemas I: diferenças entre novatos e especialistas. *Investigações em Ensino de Ciências, 1* (2), 176-192.

Cross, J. H., & Hendrix, T. D. (2006). jGRASP: a lightweight IDE with dynamic object viewers for CS1 and CS2. *SIGCSE Bulletin, 38* (3), 356-356.

Cukierman, D., & Thompson, D. M. (2007). Learning strategies sessions within the classroom in computing science university courses. *SIGCSE Bulletin, 39* (3), 341-341.

Cuny, J., & Aspray, W. (2002). Recruitment and retention of women graduate students in computer science and engineering: results of a workshop organized by the computing research association. *SIGCSE Bulletin, 34* (2), 168-174.

CUSE - Committee on Undergraduate Science Education. (1997). *Science Teaching Reconsidered: A Handbook.* National Academy Press. Washington D. C., USA.

Daly, C. (1999). RoboProf and an introductory computer programming course. *SIGCSE Bulletin, 31* (3), 155-158.

Daly, C., & Horgan, J. (2005). Patterns of plagiarism. *SIGCSE Bulletin, 37* (1), 383-387.

Daly, C., & Waldron, J. (2004). Assessing the assessment of programming ability. *SIGCSE Bulletin, 36* (1), 210-213.

Daniels, M., Berglund, A., & Petre, M. (1999). Reflections on International Projects in Undergraduate CS Education. *Computer Science Education, 9* (3), 256-267.

Davies, S. P. (1990). The nature and development of programming plans. *International Journal of Man-Machine Studies, 32* (4), 461-481.

Davis, J., & Rebelsky, S. A. (2007). Food-first computer science: starting the first course right with PB&J. *SIGCSE Bulletin, 39* (1), 372-376.

De Bello, T. C. (1990). Comparison of Eleven Major Learning Styles Models: Variables, Appropriate Populations, Validity of Instrumentation, and Research behind Them. *Journal of Reading, Writing, and Learning Disabilities International, 6* (3), 203-222.

De Vita, G. (2001). Learning Styles, Culture and Inclusive Instruction in the Multicultural Classroom: A Business and Management Perspective. *Innovations in Education and Teaching International, 38* (2), 165-174.

Decker, R., & Hirshfield, S. (1993). Top-down teaching: object-oriented programming in CS 1. *SIGCSE Bulletin, 25* (1), 270-273.

Dehnadi, S. (2006). Testing Programming Aptitude. In P. Romero, J. Good, E. A. Chaparro, & S. Bryant (Ed.), *PPIG 2006: Proceedings of the 18th Annual Workshop of the Psychology of Programming Interest Group*, (pp. 22-37). Brighton, UK.

Dehnadi, S., & Bornat, R. (2006). *The Camel Has Two Humps (working title).* Obtido em 1 de Abril de 2010, de http://www.cs.mdx.ac.uk/research/PhDArea/saeed/paper1.pdf

Delval, J. (2002). *Introdução à Prática do Método Clínico: descobrindo o pensamento das crianças.* Editora ARTMED. Porto Alegre, Brasil.

Demetriou, A. (1998). Nooplasis: 10 + 1 postulates about the formation of mind. *The Journal of the European Association for Research in Learning and Instruction (Special issue on Learning and Instruction), 8* (4), 271-287.

Denning, P. J. (2001). The profession of IT: crossing the chasm. *Communications of the ACM, 44* (4), 21-25.

Denning, P. J. (2002). Flatlined. *Communications of the ACM, 45* (6), 15-19.

Denning, P. J. (2003). Great principles of computing. *Communications of the ACM, 46* (11), 15-20.

Denning, P. J. (2004a). Programming as a Practice. *Invited speaker on the Conference on Grand Challenges In Computing: Education.* Tyneside, Newcastle, UK.

Denning, P. J. (2004b). The field of programmers myth. *Communications of the ACM, 47* (7), 15-20.

Denning, P. J., & McGettrick, A. (2005). Recentering computer science. *Communications of the ACM, 48* (11), 15-19.

Denning, P. J., & Metcalfe, R. M. (1997). *Beyond Calculation: The Next Fifty Years of Computing.* Copernicus/Springer-Verlag New York Inc.. New York, USA.

Denning, T., Griswold, W. G., Simon, B., & Wilkerson, M. (2006). Multimodal communication in the classroom: what does it mean for us? *SIGCSE Bulletin, 38* (1), 219-223.

Dershem, H. L., & Brummund, P. (1998). Tools for Web-based sorting animation. *SIGCSE Bulletin, 30* (1), 222-226.

Descartes, R. (1993). *Discours de la methode and Meditationes de prima philosophia, (1637), as quoted in Discourse on method and Meditations on first philosophy.* (D. A. Cress, Trad.) Hackett Pub. Co.. Indianapolis, USA.

Devide, J. V., Meneely, A., Ho, C.-W., Williams, L., & Devetsikiotis, M. (2008). Jazz Sangam: A Real-Time Tool for Distributed Pair Programming on a Team Development Platform. *First International Workshop on Infrastructure for Research in Collaborative Software Engineering at FSE2008.* Atlanta, GA, USA.

Dick, M., Sheard, J., Bareiss, C., Carter, J., Joyce, D., Harding, T., et al. (2003). Addressing student cheating: definitions and solutions. *SIGCSE Bulletin, 35* (2), 172-184.

Dijkstra, E. W. (1976). *A Discipline of Programming.* Prentice-Hall. Englewood Cliffs, NJ, USA.

Dijkstra, E. W. (1989). On the Cruelty of Really Teaching Computing Science. *Communications of the ACM, 32* (12), 1388-1404.

Doran, M. V., & Langan, D. D. (1995). A cognitive-based approach to introductory computer science courses: lesson learned. *SIGCSE Bulletin, 27* (1), 218-222.

Douce, C., Livingstone, D., & Orwell, J. (2005). Automatic test-based assessment of programming: A review. *Journal on Educational Resources in Computing (JERIC), 5* (3), 4.

Dreyfus, H. L., Dreyfus, S. E., & Athanasiou, T. (1986). *Mind over machine: the power of human intuition and expertise in the era of the computer.* The Free Press. New York, USA.

Dufresne, R. J. (1988). *Problem solving: learning from experts and novices.* University of Massachusetts. National Science Foundation (NSF). Washington, D.C., USA.

Dunican, E. (2002). Making The Analogy: Alternative Delivery Techniques for First Year Programming Courses. In J. Kuljis, L. Baldwin, & R. Scoble (Ed.), *PPIG 2002: Proceedings of the 14th Annual Workshop of the Psychology of Programming Interest Group*, (pp. 89-99). Brunel University, London, UK.

Dunn, R. S., & Dunn, K. J. (1978). *Teaching Students Through Their Individual Learning Styles: A Practical Approach.* Reston Publishing Division of Prentice-Hall Publishers. Reston, VA, USA.

Dunn, R., Sklar, R. I., Beaudry, J. S., & Bruno, J. (1990). Effects of Matching and Mismatching Minority Developmental College Students' Hemispheric Preferences on Mathematics Scores. *The Journal of Educational Research, 83* (5), 283-288.

Ebel, G., & Ben-Ari, M. (2006). Affective effects of program visualization. *ICER'06: Proceedings of the Second International Workshop on Computing Education Research* (pp. 1-5). ACM. Canterbury, UK.

Ebel, R. L., & Frisbie, D. A. (1986). *Essentials of Educational Measurement.* Prentice-Hall . Englewood Cliffs, NJ, USA.

Eck, D. (s.d.). *xSortLab*. Obtido em 1 de Abril de 2010, de The xSortLab Applet: http://math.hws.edu/TMCM/java/xSortLab/

Eckerdal, A., & Thuné, M. (2005). Novice Java programmers' conceptions of "object" and "class", and variation theory. *SIGCSE Bulletin, 37* (3), 89-93.

Eckerdal, A., Thuné, M., & Berglund, A. (2005). What does it take to learn 'programming thinking'? *ICER'05: Proceedings of the First International Workshop on Computing Education Research* (pp. 135-142). ACM. Seattle, WA, USA.

Edwards, H. M., Thompson, J. B., Halstead-Nussloch, R., Arnow, D., & Oliver, D. (2000). Report on the CSEET '99 Workshop: "Establishing a Distance Education Program". *Computer Science Education, 10* (1), 57-74.

Edwards, S. H. (2003a). Improving student performance by evaluating how well students test their own programs. *Journal on Educational Resources in Computing (JERIC), 3* (3), 1.

Edwards, S. H. (2003b). Rethinking computer science education from a test-first perspective. *OOPSLA'03: Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (pp. 148-155). ACM. Anaheim, CA, USA.

Edwards, S. H. (2004). Using software testing to move students from trial-and-error to reflection-in-action. *SIGCSE Bulletin, 36* (1), 26-30.

Efopoulos, V., Dagdilelis, V., Evangelidis, G., & Satratzemi, M. (2005). WIPE: a programming environment for novices. *SIGCSE Bulletin, 37* (3), 113-117.

Ehrlich, K., & Soloway, E. (1984). An empirical investigation of the tacit plan knowledge in programming. In J. C. Thomas, & M. L. Schneider (Edits.), *Human factors in computer systems* (pp. 113-133). Ablex Publishing Corp.. Norwood, NJ, USA.

Eisenstadt, M., & Brayshaw, M. (1988). The Transparent Prolog Machine (TPM): An Execution Model And Graphical Debugger For Logic Programming. *The Journal of Logic Programming, 5* (4), 277-342.

Ellis, G. P., & R., L. G. (1994). G2 - A Design Language to help novice C programmers. *Proceedings of the 2nd All Ireland Conference on Teaching of Computing.* Dublin City University, Dublin, Ireland.

Ellsworth, C. C., Fenwick, J. B., & Kurtz, B. L. (2004). The Quiver system. *SIGCSE Bulletin, 36* (1), 205-209.

Elson, D. K., & Thomas, J. (2006). *Computer Science, Hold the Computer: Designing Optimal Algorithms for Creating Peanut Butter and Jelly Sandwiches*. Obtido em 1 de Abril de 2010, de Technology Integration Partnerships, Columbia University: http://tip.columbia.edu/index.php?option=com_content&task=view&id=104

English, J. (2004). Automated assessment of GUI programs using JEWL. *SIGCSE Bulletin, 36* (3), 137-141.

Epp, S. S. (1990). *Discrete mathematics with applications.* Wadsworth Publ. Co.. Belmont, CA, USA.

Esteves, M., & Mendes, A. J. (2003). OOP-Anim, a system to support learning of basic object oriented programming concepts. *CompSysTech '03: Proceedings of the 4th International Conference Computer Systems and Technologies* (pp. 573-579). ACM. Rousse, Bulgaria.

Esteves, M., & Mendes, A. J. (2004). A simulation tool to help learning of object oriented programming basics. *FIE'2004: Proceedings of the 34th Annual Frontiers in Education, 2004*, *2*, pp. 7-12. Savannah, GA, USA.

Evangelidis, G., Dagdilelis, V., Satratzemi, M., & Efopoulos, V. (2001). X-Compiler: Yet Another Integrated Novice Programming Environment. *ICALT '01: Proceedings of the IEEE International Conference on Advanced Learning Technologies* (pp. 166-169). IEEE Computer Society. Madison, Wisconsin, USA.

Facione, N. C., Facione, P. A., & Sanchez, C. A. (1994). Critical thinking disposition as a measure of competent clinical judgment: the development of the California Critical Thinking Disposition Inventory. *Journal of Nursing Education, 33* (8), 345-350.

Facione, P. A. (1984). Toward a Theory of Critical Thinking. *Liberal Education, 70* (3), 253-261.

Facione, P. A. (1990). *Critical Thinking: A Statement of Expert Consensus for Purposes of Educational Assessment and Instruction. Research Findings and Recommendations.* Research Report, California State University, Fullerton, CA, USA.

Fagin, B., & Merkle, L. (2003). Measuring the effectiveness of robots in teaching computer science. *SIGCSE Bulletin, 35* (1), 307-311.

Farthing, D. W., Jones, D. M., & McPhee, D. (1998). Permutational multiple-choice questions: an objective and efficient alternative to essay-type examination questions. *SIGCSE Bulletin, 30* (3), 81-85.

Felder, R. M. (1993). Reaching the Second Tier: Learning and Teaching Styles in College Science Education. *Journal of College Science Teaching, 23* (5), 286-290.

Felder, R. M. (1995). A Longitudinal Study of Engineering Student Performance and Retention: IV. Instructional Methods and Student Responses to Them. *Journal of Engineering Education, 84* (4), 361-367.

Felder, R. M. (1996). Matters of Styles. *ASEE Prism, 6* (4), 18-23.

Felder, R. M., & Brent, R. (2000). Is Technology a Friend or Foe of Learning? *Chemical Engineering Education, 34* (4), 326-327.

Felder, R. M., & Brent, R. (2005). Understanding Student Differences. *Journal of Engineering Education, 94* (1), 57-72.

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

Felder, R. M., & Soloman, B. A. (s.d.). *Learning styles and strategies.* Obtido em 1 de Abril de 2010, de Richard Felder's Education-Related Publications: http://www4.ncsu.edu/unity/lockers/users/f/felder/public/ILSdir/styles.htm

Felder, R. M., & Spurlin, J. E. (2005). Applications, Reliability, and Validity of the Index of Learning Styles. *Journal of Engineering Education, 21* (1), 103-112.

Felder, R. M., Felder, G. N., & Dietz, E. J. (2002). The effects of personality type on engineering student performance and attitudes. *Journal of Engineering Education, 91* (1), 3-17.

Felder, R. M., Forrest, K. D., Baker-Ward, L., Dietz, E. J., & Mohr, P. H. (1993). A longitudinal study of engineering student performance and retention: I. Success and failure in the introductory course. *Journal of Engineering Education, 82* (1), 15-21.

Felder, R. M., Rugarcia, A., & Stice, J. E. (2000). The Future of Engineering Education: Part 5. Assessing Teaching Effectiveness and Educational Scholarship. *Chemical Engineering Education, 34* (3), 198-207.

Fenton, J., & Beck, K. (1989). Playground: an object-oriented simulation system with agent rules for children of all ages. *SIGPLAN Notices, 24* (10), 123-137.

Fincher, S. (2006). Special issue on CSE Pedagogic patterns. *Computer Science Education, 16* (2), 75-75.

Fincher, S., & Petre, M. (Edits.). (2004). *Computer Science Education Research.* Taylor & Francis. Lisse, The Netherlands.

Findler, R. B., Clement, J., Flanagan, C., Flat, M., Krishnamurth, S., Steckle, P., et al. (2002). DrScheme: A Programming Environment for Scheme. *Journal of Functional Programming, 12* (2), 159-182.

Fisher, R. (1990). *Teaching Children to Think.* Basil Blackwell. Oxford, UK.

Fix, V., & Wiedenbeck, S. (1996). An intelligent tool to aid students in learning second and subsequent programming languages. *Computers & Education, 27* (2), 71-83.

Flavell, J. H. (1976). Metacognitive aspects of problem solving. In L. B. Resnick (Ed.), *The nature of intelligence* (pp. 231-235). Lawrence Elbaum Associates. Hillsdale, NJ, USA.

Forman, G. E., & Fosnot, C. T. (1982). The use of Piaget's constructivism in early childhood education programs. In B. Spodek (Ed.), *Handbook of research in early childhood education* (pp. 185-214). Free Press. New York, USA.

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code.* Addison-Wesley Longman Publishing Co., Inc.. Boston, MA, USA.

Foxley, E. (1999). *Ceilidh documentation on the World Wide Web*. Obtido em 1 de Abril de 2010, de http://www.cs.nott.ac.uk/~ceilidh/papers.html

Freund, S. N., & Roberts, E. S. (1996). Thetis: An ANSI C programming environment designed for introductory use. *SIGCSE Bulletin, 28* (1), 300-304.

Fuller, U., Johnson, C. G., Ahoniemi, T., Cukierman, D., Hernán-Losada, I., Jackova, J., et al. (2007). Developing a computer science-specific learning taxonomy. *SIGCSE Bulletin, 39* (4), 152-170.

Gagné, E. D. (1985). *The cognitive psychology of school learning.* Little Brown and Company. Boston, MA, USA.

Gagné, R. (1965). *The conditions of learning.* Holt, Rinehart and Winston. New York, USA.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional. Boston, MA, USA.

García-Beltrán, A., Martínez, R., Jaén, J. A., Tapia, S., & Arranz, J. M. (2005). Making good use of AulaWeb in Computer Science Learning-Teaching. *ICECE'2005: Internacional Conference on Engineering and Computer Education.* Madrid, Spain.

Gayo-Avello, D., & Fernandez-Cuervo, H. (2003). Online Self-Assessment as a Learning Method. *ICALT'03: Proceedings of the Third IEEE International Conference on Advanced Learning Technologies* (p. 254). IEEE Computer Society. Athens, Greece.

GCCER. (2004). *Conference on Grand Challenges in Computing: Education*. Obtido em 1 de Abril de 2010, de http://external.cis.strath.ac.uk/educ_grand_challenges/programme.html

Gelfand, N., Goodrich, M. T., & Tamassia, R. (1998). Teaching data structure design patterns. *SIGCSE Bulletin, 30* (1), 331-335.

Gersting, J. L. (2000). Computer Science Distance Education Experience in Hawaii. *Computer Science Education, 10* (1), 95-106.

Giguette, R. (2003). Pre-games: games designed to introduce CS1 and CS2 programming assignments. *SIGCSE Bulletin, 35* (1), 288-292.

Gil Pérez, D., Martinez-Torregrosa, J., & Senent Pérez, F. (1988). El fracaso en la resolución de problemas de física: una investigación orientada por nuevos supuestos. *Enseñanza de las ciencias: revista de investigación y experiencias didácticas, 6* (2), 131-146.

Giorgetti, M. F., & Kuri, N. P. (2004). *Tipos de Personalidade e Estilos de Aprendizagem: Proposições para o Ensino de Engenharia.* Tese de Doutoramento, Universidade de São Carlos, São Carlos, SP, Brasil.

Goldstein, H. H., & von Neumann, J. (1947). *Planning and Coding Problems of an Electronic Computing Instrument.* Report prepared in accordance with the terms of contract W-36-034-ORD-7481 between the Research and Development Service, Ordnance Department, U.S. Army and the Institute for Advanced Study, Princeton, NJ, USA.

Gomes, A. (2000). *Ambiente de Suporte à Aprendizagem de Conceitos Básicos de Programação.* Dissertação de Mestrado, Departamento de Engenharia Informática, Faculdade de Ciências e Tecnologia da Universidade de Coimbra, Coimbra, Portugal.

Gomes, A., & Mendes, A. J. (2007). Learning to program - difficulties and solutions. *ICEE'07: Proceedings of the International Conference on Engineering Education, (CD-ROM).* Coimbra, Portugal.

Gomes, A., & Mendes, A. J. (2008). A study on student's characteristics and programming learning. In J. Luca, & E. R. Weippl (Ed.), *EDMEDIA'08: Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications 2008* (pp. 2895-2904). AACE. Vienna, Austria.

Gomes, A., & Mendes, A. J. (2009). Bloom's taxonomy based approach to learn basic programming. In G. Siemens, & C. Fulford (Ed.), *EDMEDIA'09: Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications 2009* (pp. 2547-2554). AACE. Honolulu, Hawaii, USA.

Gomes, A., Carmo, L., Almeida, M. E., & Mendes, A. J. (2006). Mathematics and programming problem solving. *Proceedings of the 3rd E-learning Conference on Computer Science Education.* Coimbra, Portugal.

Gries, D. (1974). What should we teach in an introductory programming course? *SIGCSE Bulletin, 6* (1), 81-89.

Gronlund, N. E. (1981). *Measurement and evaluation in teaching* (4th ed.). Macmillan. New York, USA.

Haibt, L. M. (1959). A program to draw multilevel flow charts. *IRE-AIEE-ACM '59 (Western): Proceedings of the Western Joint Computer Conference. 15*, pp. 131-137. ACM. San Francisco, California, USA.

Haladyna, T. M. (1999). *Developing and Validating Multiple-Choice Questions* (2nd ed.). Lawrence Erlbaum Associates. Mahwah, NJ, USA.

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

Hardiman, P. H., Dufresne, R. J., & Mestre, J. P. (1989). The relation between problem categorization and problem solving among novices and experts. *Memory & Cognition, 17* (5), 627-638.

Harlan, R. M., Levine, D. B., & McClarigan, S. (2001). The Khepera robot and the kRobot class: a platform for introducing robotics in the undergraduate curriculum. *SIGCSE Bulletin, 33* (1), 105-109.

Hayes, J. R. (1981). *The complete problem solver.* Franklin Institute Press. Philadelphia, PA, USA.

Heller, J. I., & Reif, F. (1984). Prescribing Effective Human Problem-Solving Processes: Problem Description in Physics. *Cognition and Instruction, 1* (2), 177-216.

Henriksen, P., & Kölling, M. (2004). greenfoot: Combining Object Visualisation with Interaction. *OOPSLA '04: Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications* (pp. 73-82). ACM. Vancouver, BC, Canada.

Hernán-Losada, I., Lázaro-Carrascosa, C., & Velázquez-Iturbide, J. Á. (2004). On the use of Bloom's taxonomy as a basis to design educational software on programming. *WCETE'2004: Proceedings of the World Conference on Engineering and Technology Education* (pp. 351-355). COPEC. Brasil.

Hernán-Losada, I., Velázquez-Iturbide, J. Á., & Lázaro-Carrascosa, C. (2006). Programming learning tools based on Bloom's taxonomy: proposal and accomplishments. *SIIE 2006: Proceedings of the VIII International Symposium of Computers in Education*, (pp. 325-334). León, España.

Higgins, C., Hegazy, T., Symeonidis, P., & Tsintsifas, A. (2003). The CourseMarker CBA System: Improvements over Ceilidh. *Education and Information Technologies, 8* (3), 287-304.

Hood, C. S., & Hood, D. J. (2005). Teaching programming and language concepts using LEGOs®. *SIGCSE Bulletin, 37* (3), 19-23.

Hooper, C., Carr, L., Davis, H., Millard, D., White, S., & Wills, G. (2007). AnnAnn and AnnAnn.Net : Tools for Teaching Programming. *Journal of Computers, 2* (5), 9-16.

Howard, R. A., Carver, C. A., & Lane, W. D. (1996). Felder's learning styles, Bloom's taxonomy, and the Kolb learning cycle: tying it all together in the CS2 course. *SIGCSE Bulletin, 28* (1), 227-231.

Hueras, J., & Ledgard, H. (1977). An automatic formatting program for PASCAL. *SIGPLAN Notices, 12* (7), 82-84.

Huet, I., & Tavares, J. (2004). *A qualidade do ensino nas Universidades: estudo de caso.* Obtido em 1 de Abril de 2010, de Laboratório de Estudo e Intervenção no Ensino Superior (LEIES) da Universidade de Aveiro: http://www2.dce.ua.pt/LEIES/daes_qualidadensino.pdf

Huitt, W. (1992). Problem solving and decision making: considerations of individual differences using the Myers-Briggs Type Indicator. *Journal of Psychological Type, 24*, 33-44.

Huitt, W., & Hummel, J. (2003). *Piaget's Theory of Cognitive Development.* Obtido em 1 de Abril de 2010, de Educational Psychology Interactive: http://www.edpsycinteractive.org/topics/cogsys/piaget.html

Hundhausen, C. D., Brown, J. L., Farley, S., & Skarpas, D. (2006). A methodology for analyzing the temporal evolution of novice programs based on semantic components. *ICER'06: Proceedings of the Second International Workshop on Computing Education Research* (pp. 59-71). ACM. Canterbury, UK.

Hundhausen, C. D., Douglas, S. A., & Stasko, J. T. (2002). A Meta-Study of Algorithm Visualization Effectiveness. *Journal of Visual Languages & Computing, 13* (3), 259-290.

Hvorecky, J. (1992). Karel the Robot for PC. In P. Brusilovsky, & V. Stefanuk (Ed.), *Proceedings of East-West Conference on Emerging Computer Technologies in Education*, (pp. 157-160). Moscow, Russia.

Hyman, R., & Anderson, B. (1965). Solving Problems. *International Science and Technology*, (pp. 36-41).

Hyvönen, J., & Malmi, L. (1993). TRAKLA - A System for Teaching Algorithms Using Email and a Graphical Editor. *Proceedings of the HYPERMEDIA'93*, (pp. 141-147). Vaasa, Finland.

Ihantola, P., Karavirta, V., Korhonen, A., & Nikander, J. (2005). Taxonomy of effortless creation of algorithm visualizations. *ICER '05: Proceedings of the First International Workshop on Computing Education Research* (pp. 123-133). ACM. Seattle, WA, USA.

Imberman, S. P., & Klibaner, R. (2005). A robotics lab for CS1. *Journal of Computing Sciences in Colleges, 21* (2), 131-137.

Ingargiola, G., Hoskin, N., Aiken, R., Dubey, R., Wilson, J., Papalaskari, M.-A., et al. (1994). A repository that supports teaching and cooperation in the introductory AI course. *SIGCSE Bulletin, 26* (1), 36-40.

Jackson, D., & Usher, M. (1997). Grading student programs using ASSYST. *SIGCSE Bulletin, 29* (1), 335-339.

Jadud, M. C. (2006). Methods and tools for exploring novice compilation behaviour. *ICER'06: Proceedings of the Second International Workshop on Computing Education Research* (pp. 73-84). ACM. Canterbury, UK.

James, W. B., & Blank, W. E. (1993). Review and Critique of Available Learning-Style Instruments for Adults. *New Directions for Adult and Continuing Education, 59*, 47-57.

Janzen, D. S., & Saiedian, H. (2006). Test-driven learning: intrinsic integration of testing into the CS/SE curriculum. *SIGCSE Bulletin, 38* (1), 254-258.

Jeffries, R., Turner, A. A., Polson, P. G., & Atwood, M. E. (1980). The processes involved in designing software. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 255-283). Lawrence Erlbaum Associates, Inc.. Hillsdale, NJ, USA.

Jehng, J.-C. J., Shih, Y.-F., Liang, S., & Chan, T.-W. (1994). TurtleGraph: A Computer Supported Cooperative Learning Environment. *Proceedings of ED-MEDIA'94 - World Conference on Educational Multimedia and Hypermedia*, (pp. 293-298). Vancouver, British Columbia, Canada.

Jenkins, T. (2002). On the Difficulty of Learning to Program. *Proceedings of the 3rd Annual LTSN-ICS Conference* (pp. 53-58). The Higher Education Academy. Loughborough University, UK.

Johnson, C. G., & Fuller, U. (2006). Is Bloom's taxonomy appropriate for computer science? *Baltic Sea'06: Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006* (pp. 120-123). ACM. Uppsala, Sweden.

Johnson, W. L., & Soloway, E. (1985). PROUST: an automatic debugger for PASCAL programs. *BYTE, 10* (4), 179-190.

Johnson, W. L., Draper, S. W., & Soloway, E. (1983). An effective bug classification scheme must take the programmer into account. *Proceedings of the Workshop on High-Level Debugging.* Palo Alto, CA, USA.

Jonassen, D. H., Peck, K. L., & Wilson, B. G. (1999). *Learning With Technology: A Constructivist Perspective.* Prentice Hall. Upper Saddle River, NJ, USA.

Jones, C. G. (2004). Test-driven development goes to school. *Journal of Computing Sciences in Colleges, 20* (1), 220-231.

Jung, C. G. (1971). *Psychological Types.* Princeton University Press. Princeton, NJ, USA.

Kagan, J. (1965). Reflection-Impulsivity and Reading Ability in Primary Grade Children. *Child Development, 36* (3), 609-628.

Kagan, J., & Lewis, M. (1965). Studies of Attention in the Human Infant. *Merrill-Palmer Quarterly, 11*, 95–127.

Karavirta, V., Korhonen, A., & Malmi, L. (2006). On the Use of Resubmissions in Automatic Assessment Systems. *Computer Science Education, 16* (3), 229-240.

Kay, J., & Tyler, P. (1992). A Microworld for Developing Learning Design Strategies. *Computer Science Education, 3* (2), 111-122.

Keefe, J. W. (1982). Assessing students learning styles: An overview. *Students Learning Styles and Brain Behaviour*, 43-57.

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR), 37* (2), 83-137.

Kelleher, C., Cosgrove, D., Culyba, D., Forlines, C., Pratt, J., & Pausch, R. (2002). Alice2: Programming without Syntax Errors. *Short Paper and Demonstration at the 2002 Conference on User Interface Software and Technology.* Paris, France.

Kessler, C. M., & Anderson, J. R. (1989). Learning flow of control: Recursive and iterative procedures. In J. C. Spohrer, & E. I. Soloway (Edits.), *Studying the novice programmer* (pp. 229–260). Lawrence Erlbaum Associates. Hillsdale, NJ, USA.

Kim, J., & Lerch, F. J. (1997). Why Is Programming (Sometimes) So Difficult? Programming as Scientific Discovery in Multiple Problem Spaces. *Information Systems Research, 8* (1), 25-50.

King, P. M., & Kitchener, K. S. (1994). *Developing Reflective Judgment: Understanding and Promoting Intellectual Growth and Critical Thinking in Adolescents and Adults.* Jossey-Bass. San Francisco, CA, USA.

Kizlik, B. (2010). *Thinking Skills Vocabulary and Definitions*. Obtido em 1 de Abril de 2010, de Education Information for New and Future Teachers: http://www.adprima.com/thinkskl.htm

Kline, P. (2000). *Handbook of Psychological Testing* (2nd ed.). Routledge. New York, USA.

Knowlton, J. (1966). On the definition of "Picture". *AV Communications Review, 14* (2), 157-183.

Knuth, D. E. (1963). Computer-drawn flowcharts. *Communications of the ACM, 6* (9), 555-563.

Kobayashi, K., Uchida, Y., & Watanabe, K. (2003). A study of battle strategy for the Robocode. *Prooceedings of the SICE 2003 Annual Conference*, *3*, pp. 3373 - 3376. Fukui, Japan.

Koile, K., & Singer, D. (2006). Improving learning in CS1 via tablet-PC-based in-class assessment. *ICER '06: Proceedings of the Second International Workshop on Computing Education Research* (pp. 119-126). ACM. Canterbury, UK.

Kolb, D. A. (1984). *Experiential Learning: Experience as the Source of Learning and Development.* Prentice-Hall. Englewood Cliffs, NJ, USA.

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

Kolb, D. A. (1997). A gestão e o processo de aprendizagem. In K. Starkey, *Como as organizações aprendem: relatos dos sucessos das grandes empresas* (pp. 321-341). Futura. São Paulo, Brasil.

Kölling, M. (2009). *Quality-oriented teaching of programming*. Obtido em 1 de Abril de 2010, de mik's blog - Thoughts on Programming Education: http://blogs.kent.ac.uk/mik/2009/09/04/quality-oriented-teaching-of-programming/

Kölling, M., & Henriksen, P. (2005). Game programming in introductory courses with direct state manipulation. *SIGCSE Bulletin, 37* (3), 59-63.

Kölling, M., Quig, B., Patterson, A., & Rosenberg, J. (2003). The BlueJ system and its pedagogy. *Journal of Computer Science Education, Special issue on Learning and Teaching Object Technology, 13* (4), 249-268.

Korhonen, A., Malmi, L., & Silvasti, P. (2003). TRAKLA2: a framework for automatically assessed visual algorithm simulation exercises. *Proceedings of Third Annual Baltic Conference on Computer Science Education*, (pp. 48–56). Joensuu, Finland.

Korhonen, A., Malmi, L., Myllyselkä, P., & Scheinin, P. (2002). Does it make a difference if students exercise on the web or in the classroom? *SIGCSE Bulletin, 34* (3), 121-124.

Kramer, J. (2007). Is abstraction the key to computing? *Communications of the ACM, 50* (4), 36-42.

Krathwohl, D. R. (2002). A Revision of Bloom's Taxonomy: An Overview. *Theory Into Practice, 41* (4), 212-218.

Krathwohl, D. R., Bloom, B. S., & Masia, B. (1964). *Taxonomy of Educational Objectives: The Classification of Educational Goals - Handbook 2: Affective Domain* (1 ed.). Longman. London, UK.

Kumar, A. N. (2003). Learning Programming by Solving Problems. In L. N. Cassel, & R. A. Reis (Edits.), *Informatics Curricula and Teaching Methods* (Vol. 245, pp. 29-39). Kluwer Academic. Florianópolis, SC, Brasil.

Kumar, A. N. (2005a). Generation of problems, answers, grade, and feedback - Case study of a fully automated tutor. *Journal on Educational Resources in Computing (JERIC), 5* (3), 3.

Kumar, A. N. (2005b). Results from the evaluation of the effectiveness of an online tutor on expression evaluation. *SIGCSE Bulletin, 37* (1), 216-220.

Kundratova, M., & Turek, I. (2001). *Chapters from engineering pedagogy. Educational Objectives (in Slovak).* STU Bratislava. Bratislava, Slovakia.

Kuri, N. P. (2004). *Tipos de personalidade e estilos de aprendizagem: proposições para o ensino de engenharia.* Tese de Doutoramento, Centro de Ciências Exatas e Tecnologia, Universidade Federal de São Carlos, São Carlos, Brasil.

Kuri, N. P., & Truzzi, O. M. (2002). Learning Styles of Freshmen Engineering Students. *Proceedings of the 2002 International Conference on Engineering Education.* International Network for Engineering Education and Research. Arlington, VA, USA.

Lahti, A. M. (1956). The Inductive-Deductive Method and the Physical Science Laboratory. *The Journal of Experimental Education, 24* (3), 149-163.

Lahtinen, E. (2007). A Categorization of Novice Programmers: A Cluster Analysis Study. In J. Sajaniemi, M. Tukiainen, R. Bednarik, & S. Nevalainen (Ed.), *PPIG 2007: Proceedings of the 19th Annual Workshop of the Psychology of Programming Interest Group*, (pp. 32-41). Joensuu, Finland.

Lahtinen, E., & Ahoniemi, T. (2005). Visualizations to Support Programming on Different Levels of Cognitive Development. *Proceedings of the Fifth Koli Calling Conference on Computer Science Education*, (pp. 87-94). Koli, Finland.

Lahtinen, E., Ala-Mutka, K., & Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. *SIGCSE Bulletin, 37* (3), 14-18.

Lahtinen, S., Sutinen, E., & Tarhio, J. (1998). Automated Animation of Algorithms with Eliot. *Journal of Visual Languages & Computing, 9* (3), 337-349.

Lancaster, T., & Culwin, F. (2004). A Comparison of Source Code Plagiarism Detection Engines. *Computer Science Education, 14* (2), 101-112.

Lane, H. C., & VanLehn, K. (2003). Coached program planning: dialogue-based support for novice program design. *SIGCSE Bulletin, 35* (1), 148-152.

Larkin, J. (1981). Enriching Formal Knowledge: A Model for Learning to Solve Textbook Physics Problems. In J. R. Anderson (Ed.), *Cognitive Skills and Their Acquisition* (pp. 311-334). Lawrence Erlbaum Associates, Inc.. Hillsdale, NJ, USA.

Larkin, J., & Reif, F. (1979). Understanding and Teaching Problem-Solving in Physics. *International Journal of Science Education, 1* (2), 191-203.

Larkin, J., McDermott, J., Simon, D. P., & Simon, H. A. (1980). Expert and Novice Performance in Solving Physics Problems. *Science, 208* (4450), 1335-1342.

Lawhead, P. B., Bland, C. G., Barnes, D. J., Duncan, M. E., Goldweber, M., Hollingsworth, R. G., et al. (2003). A Road Map for Teaching Introductory Programming Using LEGO Mindstorms Robots. *SIGCSE Bulletin, 35* (2), 191-201.

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

Lewandowski, G., & Morehead, A. (1998). Computer science through the eyes of dead monkeys: learning styles and interaction in CS I. *SIGCSE Bulletin, 30* (1), 312-316.

Lewin, K. (1936). *Principles of topological psychology.* Mc-Graw-Hill Book Company, Inc.. New York, USA.

Lindholm, M. (2005). Development of object-understanding among students in the humanities. *SIGCSE Bulletin, 37* (3), 382-382.

Lindholm, M. (2007). Conceptions of Object-oriented Terms: A Study in Progress. *Proceedings of the 2007 Work-in-Progress Workshop of the Psychology of Programming Interest Group.* Salford, UK.

Linn, M. C., & Clancy, M. J. (1992). The case for case studies of programming problems. *Communications of the ACM, 35* (3), 121-132.

Linn, M. C., & Dalbey, J. (1985). Cognitive consequences of Programming Instruction: Instruction, Access, and Ability. *Educational Psychologist, 20* (4), 191-206.

Linn, R. L., & Gronlund, N. E. (1995). *Measurement and Assessment in Teaching* (7th ed.). Prentice Hall. Upper Saddle River, NJ, USA.

Lister, R. (2000). On blooming first year programming, and its blooming assessment. *ACSE'00: Proceedings of the Australasian Conference on Computing Education* (pp. 158-162). ACM. Melbourne, Australia.

Lister, R. (2003). A research manifesto, and the relevance of phenomenography. *SIGCSE Bulletin, 35* (2), 15-16.

Lister, R., & Leaney, J. (2003a). First year programming: let all the flowers bloom. *ACE'03: Proceedings of the Fifth Australasian Conference on Computing Education* (pp. 221-230). Australian Computer Society, Inc.. Adelaide, Australia.

Lister, R., & Leaney, J. (2003b). Introductory programming, criterion-referencing, and bloom. *SIGCSE Bulletin, 35* (1), 143-147.

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., et al. (2004). A multi-national study of reading and tracing skills in novice programmers. *SIGCSE Bulletin, 36* (4), 119-150.

Lister, R., Simon, B., Thompson, E., Whalley, J. L., & Prasad, C. (2006). Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. *SIGCSE Bulletin, 38* (3), 118-122.

Litzinger, T. A., Lee, S. H., Wise, J. C., & Felder, R. M. (2007). A Psychometric Study of the Index of Learning Styles. *Journal of Engineering Education, 96* (4), 309-319.

Livesay, G. A., Dee, K. C., Nauman, E. A., & Hites, J. L. (2002). Engineering student learning styles: a statistical analysis using Felder's Index of Learning Styles. *Proceedings of the 2002 Conference of the American Society for Engineering Education.* Montreal, Quebec, Canada.

Lopes, W. M. (2002). *ILS—Inventário de Estilos de Aprendizagem de Felder-Soloman: Investigação de sua Validade em Estudantes Universitários de Belo Horizonte.* Tese de Mestrado, Universidade Federal de Santa Catarina, Florianópolis,Santa Catarina,Brasil.

Luck, M., & Joy, M. (1999). A Secure On-Line Submission System. *Software - Practice and Experience, 29* (8), 721-740.

Magill, R. A. (1980). *Motor learning: Concepts and applications.* W. C. Brown Co.. Dubuque, Iowa, USA.

Malmi, L., & Korhonen, A. (2008). Active Learning and Examination Methods in a Data Structures and Algorithms Course. In J. Bennedsen, M. E. Caspersen, & M. Kölling (Edits.), *Reflections on the Teaching of Programming: Methods and Implementations* (pp. 210-227). Springer-Verlag. Berlin, Germany.

Malmi, L., Karavirta, V., Korhonen, A., & Nikander, J. (2005). Experiences on automatically assessed algorithm simulation exercises with different resubmission policies. *Journal on Educational Resources in Computing (JERIC), 5* (3), 7.

Malpohl, G. (1996). *JPlag - Detecting Software Plagiarism*. Obtido em 1 de Abril de 2010, de Institute for Program Structures and Data Organization: https://www.ipd.uni-karlsruhe.de/jplag/

Manaris, B., & McCauley, R. (2004). Incorporating HCI into the undergraduate curriculum: Bloom's taxonomy meets the CC'01 curricular guidelines. *FIE 2004: Proceedings of the 34th ASEE/IEEE Annual Frontiers in Education*, *1*, pp. T2H/10 - T2H/15. Savannah, GA, USA.

Marcelino, M., Mihaylov, T., & Mendes, A. J. (2008). H-SICAS, a handheld algorithm animation and simulation tool to support initial programming learning. *Proceedings of the 38th ASEE/IEEE Frontiers in Education Conference* (pp. T4A/7-T4A/12). IEEE. Saratoga Springs, NY, USA.

Margolis, J., & Fisher, A. (2003). *Unlocking the Clubhouse : Women in Computing.* The MIT Press. Cambridge, MA, USA.

Martin, R. C. (2003). *Agile Software Development: Principles, Patterns, and Practices.* Prentice Hall. Upper Saddle River, NJ, USA.

Marton, F., & Booth, S. (1997). *Learning and Awareness.* Lawrence Erlbaum Associates. Mahwah, NJ, USA.

Marton, F., & Säljö, R. (1997). Approaches to Learning. In F. Marton, D. Hounsell, & N. Entwistle (Edits.), *The Experience of Learning* (2nd ed., pp. 39-58). Scottish Academic Press. Edinburgh, UK.

Mayer, R. E. (1981). The Psychology of How Novices Learn Computer Programming. *ACM Computing Surveys (CSUR), 13* (1), 121-141.

Mayer, R. E. (1998). Cognitive, metacognitive and motivational aspects of problem solving. *Instructional Science, 26* (1-2), 49-63.

Mayer, R. E., Dyck, J. L., & Vilberg, W. (1986). Learning to program and learning to think: what's the connection? *Communications of the ACM, 29* (7), 605-610.

McConnell, T. (1934). Discovery Versus Authoritative Identification in the Learning of Children. *Studies in Education, 9* (5), 13-60.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., et al. (2001). A multinational, multi-institutional study of assessment of programming skills of first-year CS students. *SIGCSE Bulletin, 33* (4), 125-140.

McGettrick, A., Boyle, R., Ibbett, R., Lloyd, J., Lovegrove, G., & Mander, K. (2005). Grand Challenges in Computing: Education–A Summary. *The Computer Journal, 48* (1), 42-48.

McKeachie, W. J. (1995). Learning styles can become learning strategies. *The National Teaching and Learning Forum, 4* (6), 1-3.

Mead, J., Gray, S., Hamer, J., James, R., Sorva, J., Clair, C. S., et al. (2006). A cognitive approach to identifying measurable milestones for programming skill acquisition. *SIGCSE Bulletin, 38* (4), 182-194.

Mendes, A. J., & Mendes, T. (1988). VIP - A tool to visualize programming examples. *Proceedings of the EACT 88—Education and Application of Computer Technology.* Malta.

Merrill, M. D. (1994). Lesson segments based on component display theory. In M. D. Merrill (Ed.), *Instructional design theory* (pp. 177-212). Educational Technology Publications. Englewood Cliffs, NJ, USA.

Merrill, M. D. (1994). The descriptive component display theory. In M. D. Merrill (Ed.), *Instructional design theory* (pp. 111-158). Educational Technology Publications. Englewood Cliffs, NJ, USA.

Merrill, M. D. (1994). The prescriptive component display theory. In M. D. Merrill (Ed.), *Instructional design theory* (pp. 159-176). Educational Technology Publications. Englewood Cliffs, NJ, USA.

Messick, S. (1969). *The Criterion Problem in the Evaluation of Instruction: Assessing Possible Not Just Intended Outcomes.* CSE Report, 370 (22), University of California, Los Angeles, CA, USA.

Messick, S. (1984). The nature of cognitive styles: Problems and promise in educational practice. *Educational Psychologist, 19* (2), 59-74.

Mestre, J. P. (2001). Implications of research on learning for the education of prospective science and physics teachers. *Physics Education, 36* (1), 44-51.

Meyer, B. (1993). Toward an object-oriented curriculum. *Journal of Object Oriented Programming: Education & Training*, 76-81.

Miller, P., Pane, J., Meter, G., & Vorthmann, S. (1994). Evolution of Novice Programming Environments: The Structure Editors of Carnegie Mellon University. *Interactive Learning Environments, 4* (2), 140-158.

Miyadera, Y., Huang, N., & Yokoyama, S. (2000). A programming language education system based on program animation. *IFIP 16th World Computer Congress: Proceedings of the Education Uses of Information and Communication Technologies World Computer Congress*, (pp. 258-261). Beijing, China.

Monroy, F. J. (2010). *Proposal for evaluating computer programming algorithms to provide instructional guidance and give advice.* PhD Thesis, Universidad de Castilla-La Mancha, España.

Montgomery, S. M. (1995). Addressing diverse learning styles through the use of multimedia. *FIE '95: Proceedings of the 25th ASEE/IEEE Frontiers in Education Conference. 1*, pp. 3a2.13-3a2.21. IEEE Computer Society. Atlanta, GA, USA.

Moon, J. A. (2002). *How to use level descriptors.* SEEC Office, University of East London. Southern England Consortium for Credit Accumulation and Transfer SEEC. London, UK.

Moreno, A., Myller, N., Ben-Ari, M., & Sutinen, E. (2004). Program animation in jeliot 3. *SIGCSE Bulletin, 36* (3), 265-265.

Morgan, G., Stephanou, A., & Simpson, B. (2000). *Aptitude Profile Test Series: Manual*. Obtido em 1 de Abril de 2010, de Australian Council for Educational Research: http://www.acer.edu.au

Morris, D. S. (2003). Automatic grading of student's programming assignments: an interactive process and suite of programs. *FIE'03: Proceedings of the 33rd ASEE/IEEE Frontiers in Education Conference. 3*, pp. S3F.1 - S3F.5. IEEE Computer Society. Boulder, CO, USA.

Motil, J., & Epstein, D. (2000). *JJ: a Language Designed for Beginners (Less Is More).* Obtido em 1 de Abril de 2010, de http://www.ecs.csun.edu/~jmotil/TeachingWithJJ.pdf

Mourtos, N. J., & Allen, E. L. (2000). Assessing the effectiveness of a faculty instructional development program, part 1: Cooperative Learning. *Proceedings of the 2nd Global Congress on Engineering Education.* Wismar, Germany.

Mourtos, N. J., & Allen, E. L. (2003). Assessing the effectiveness of a faculty instructional development program, part 2: teaching and learning styles. *Proceedings of the 6th UICEE Annual Conference on Engineering Education.* Cairns, Queensland, Australia.

Mueller, F., & Hosking, A. L. (2003). Penumbra: an Eclipse plugin for introductory programming. *Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology eXchange*, (pp. 65-68). Anaheim, CA, USA.

Myers, B. A. (1980). *Displaying Data Structures for Interactive Debugging.* Technical Report CSL-80-7, XEROX Palo Alto Research Center, Palo Alto, CA, USA.

Myers, B. A. (1983). INCENSE: A system for displaying data structures. *SIGGRAPH Computer Graphics, 17* (3), 115-125.

Myers, B. A. (1986). Visual Programming, Programming by Example, and Program Visualization: A Taxonomy. *CHI'86: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 59-66). ACM. Boston, MA, USA.

Myers, B. A. (1990). Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing, 1* (1), 97-123.

Myers, B. A., Chandhok, R., & Sareen, A. (1988). Automatic Data Visualization for Novice Pascal Programmers. *Proceedings of The IEEE Workshop on Visual Languages* (pp. 192-198). IEEE Computer Society Press. Pittsburgh, PA, USA.

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

Myers, I. B., & Myers, P. B. (1980). *Gifts Differing: Understanding Personality Type* (reprinted 1995 ed.). Davies-Black Publishing. Boston, MA, USA.

Naps, T. L. (1990). Algorithm visualization in computer science laboratories. *SIGCSE Bulletin, 22* (1), 105-110.

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

Naps, T. L., Bergin, J., Jiménez-Peris, R., McNally, M. F., Patiño-Martínez, M., Proulx, V. K., et al. (1997). Using the WWW as the delivery mechanism for interactive, visualization-based instructional modules (Report of the ITiCSE '97 Working Group on Visualization). *The Supplemental Proceedings of the Conference on Integrating Technology into Computer Science Education: Working Group Reports and Supplemental Proceedings*, (pp. 13-26). Uppsala, Sweden.

Naps, T. L., Cooper, S., Koldehofe, B., Leska, C., Rößling, G., Dann, W., et al. (2003). Evaluating the educational impact of visualization. *SIGCSE Bulletin, 35* (4), 124-136.

Naps, T. L., Eagan, J. R., & Norton, L. L. (2000). JHAVÉ - An environment to actively engage students in Web-based algorithm visualizations. *SIGCSE Bulletin, 32* (1), 109-113.

Nassi, I., & Schneiderman, B. (1973). Flowchart techniques for structured programming. *SIGPLAN Notices, 8* (8), 12-26.

Neto, F. A., Castro, T. H., & Júnior, A. N. (2006). Utilizando o Método Clínico Piagetiano para Acompanhar a Aprendizagem de Programação. *SBIE2006: XVII Simpósio Brasileiro de Informática na Educação* (pp. 527-536). Gráfica e Editora Positiva Lda. Brasília, Brasil.

Neves, D. M., & Anderson, J. R. (1981). Knowledge compilation: Mechanisms for the automatization of cognitive skills. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 57-84). Lawrence Erlbaum Associates. Hillsdale, NJ, USA.

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

Nguyen, D., & Wong, S. B. (1999). Patterns for decoupling data structures and algorithms. *SIGCSE Bulletin, 31* (1), 87-91.

Norman, D. A., & Spohrer, J. C. (1996). Learner-centered education. *Communications of the ACM, 39* (4), 24-27.

O'Brien, T., Bernold, L. E., & Akroyd, D. (1998). Myers-Briggs Type Indicator and Academic Achievement in Engineering Education. *The International Journal of Engineering Education, 14* (5), 311-315.

Ochse, R. (1990). *Before the gates of excellence : the determinants of creative genius.* Cambridge University Press. Cambridge, UK.

OECD, Organisation for Economic Co-operation and Development. (2003). *Learning for tomorrow's world. First results from PISA 2003.* Obtido em 1 de Abril de 2010, de http://www.pisa.oecd.org/dataoecd/38/30/33707234.pdf

Olimpo, G. (1988). The Robot Brothers: an environment for learning parallel programming oriented to computer education. *Computers & Education, 12* (1), 113-118.

Olimpo, G., Persico, D., Sarti, L., & Tavella, M. (1985). An experiment in introducing the basic concepts of informatics. In K. Dunkan, & D. Harris (Ed.), *WCCE'85: Proceedings of the Fourth World Conference on Computers in Education*, (pp. 31-38). Amsterdam, The Netherlands.

Oliver, D., Dobele, T., Greber, M., & Roberts, T. (2004). This course has a Bloom Rating of 3.9. *ACE'04: Proceedings of the Sixth Conference on Australasian Computing Education. 30*, pp. 227-231. Australian Computer Society, Inc.. Dunedin, New Zealand.

Ollis, C. D. (1995). *Computing With Peanut Butter And Jelly Sandwiches.* Obtido em 1 de Abril de 2010, de The Educator's Reference Desk: http://www.eduref.org/Virtual/Lessons/Science/Process_Skills/SPS0013.html

Oxford, R., & Shearin, J. (1994). Language Learning Motivation: Expanding the Theoretical Framework. *The Modern Language Journal, 78* (1), 12-28.

Pacheco, A., Henriques, J., Almeida, A. M., & Mendes, A. J. (2008). A study on basic mathematics knowledge for the enhancement of programming learning skills. *Proceedings of the IEEIII08 - Informatics Education Europe III.* Venice, Italia.

Palormo, J. M., & Fisher, B. M. (1964). *Computer Programmer Aptitude Battery (CPAB).* Science Research Associates, Inc.. Chicago, IL, USA.

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

Paterson, K. G. (1999). Student Perceptions of Internet-Based Learning Tools in Environmental Engineering Education. *Journal of Engineering Education, 88* (3), 295-304.

Pattis, R. E. (1981). *Karel the Robot: A Gentle Introduction to the Art of Programming* (2nd ed.). John Wiley & Sons, Inc.. New York, USA.

Pattis, R. E. (1990). A philosophy and example of CS-1 programming projects. *SIGCSE Bulletin, 22* (1), 34-39.

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology, 2* (2), 137-168.

Pecinovský, R., Pavlíčková, J., & Pavlíček, L. (2006). Let's modify the objects-first approach into design-patterns-first. *SIGCSE Bulletin, 38* (3), 188-192.

Pedroni, M., & Meyer, B. (2006). The inverted curriculum in practice. *SIGCSE Bulletin, 38* (1), 481-485.

Pennings, A. H., & Span, P. (1991). Estilos cognitivos e estilos de aprendizagem. In L. Almeida (Ed.), *Cognição e aprendizagem escolar.* APPORT. Porto, Portugal.

Perales, F. J. (1993). La resolución de problemas: una revisión estructurada. *Enseñanza de las Ciencias, 11* (2), 170-178.

Perkins, D. N., & Martin, F. (1986). Fragile knowledge and neglected strategies in novice programmers. In E. Soloway, & S. Iyengar (Ed.), *Papers presented at the First Workshop on Empirical Studies of Programmers on Empirical Studies of Programmers* (pp. 213-229). Ablex Publishing Corp.. Washington, D.C., USA.

Perkins, D. N., Schwartz, S., & Simmon, R. (1988). Instructional Strategies for the Problems of Novice Programmers. In R. E. Mayer (Ed.), *Teaching and Learning Computer Programming* (pp. 153-178). Lawrence Erlbaum Associates. Hillsdale, NJ, USA.

Perry, W. G. (1970). *Forms of Intellectual and Ethical Development in the College Years: A Scheme.* Holt, Rinehart, and Winston, Inc.. New York, USA.

Perry, W. G. (1988). Different worlds in the same classroom. In P. Ramsden (Ed.), *Improving learning: new perspectives* (pp. 145-161). Kogan Page Ltd. London, UK.

Petre, M., & Winder, R. (1988). Issues governing the suitability of programming languages for programming tasks. *Proceedings of the Fourth Conference of the British Computer Society on People and Computers IV* (pp. 199-215). Cambridge University Press. University of Manchester, UK.

Piaget, J. (1970). *Science of Education and the Psychology of the Child.* (D. Coltman, Trad.) Orion Press. New York, USA.

Piaget, J., & Inhelder, B. (1969). *The psychology of the child.* (H. Weaver, Trad.) Basic Books. New York, USA.

Pillay, N. (2002). Using genetic programming for the induction of novice procedural programming solution algorithms. *SAC'02: Proceedings of the 2002 ACM Symposium on Applied Computing* (pp. 578-583). ACM Press. New York, USA.

Polanyi, M. (1958). *Personal knowledge : towards a post-critical philosophy.* Univiversity of Chicago Press. Chicago, IL, USA.

Polya, G. (1945). *How to solve it: A new aspect of mathematical method.* Princeton University Press. Princeton, NJ, USA.

Prechelt, L., Malpohl, G., & Philippsen, M. (2002). Finding Plagiarisms among a Set of Programs with JPlag. *Journal of Universal Computer Science, 8* (11), 1016-1038.

Preiss, B. R. (1999). Design patterns for the data structures and algorithms course. *SIGCSE Bulletin, 31* (1), 95-99.

Pretz, J. E., Naples, A. J., & Sternberg, R. J. (2003). Recognizing, Defining and Representing Problems. In J. E. Davidson, & R. J. Sternberg (Edits.), *The psychology of problem solving* (pp. 3-30). Cambridge University Press. New York, USA.

PVW. (2000). *Program Visualization Workshop*. Obtido em 1 de Abril de 2010, de University of Joensuu: http://www.cs.joensuu.fi/pages/pvw/workshop.htm

PVW. (2002). *Second Program Visualization Workshop*. Obtido em 1 de Abril de 2010, de The Department of Science Teaching of the Weizmann Institute of Science: http://stwww.weizmann.ac.il/g-cs/benari/pvw/pvw.html

PVW. (2004). *Third Program Visualization Workshop*. Obtido em 1 de Abril de 2010, de Department of Computer Science of the University of Warwick: http://www.dcs.warwick.ac.uk/pvw04/

PVW. (2006). *Fourth Program Visualization Workshop*. Obtido em 1 de Abril de 2010, de http://www.algoanim.net/pvw2006/

Rademacher, R. A. (1999). Applying Bloom's taxonomy of cognition to knowledge management systems. *SIGCPR'99: Proceedings of the 1999 ACM SIGCPR conference on Computer personnel research* (pp. 276-278). ACM. New Orleans, LA, USA.

Rajaravivarma, R. (2005). A games-based approach for teaching the introductory programming course. *SIGCSE Bulletin, 37* (4), 98-102.

Ramadhan, H. (1992). An intelligent discovery programming system. *SAC'92: Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing: Technological Challenges of the 1990's* (pp. 149-159). ACM. Kansas City, MO, USA.

Ramsden, P. (1992). *Learning to Teach in Higher Education.* Routledge. London, UK.

Ramsden, P., & Entwistle, N. J. (1981). Effects of Academic Departments on Students' Approaches to Studying. *British Journal of Educational Psychology, 51*, 368-383.

Rapaport, W. J. (2008). *William Perry's Scheme of Intellectual and Ethical Development.* Obtido em 1 de Abril de 2010, de http://www.cse.buffalo.edu/~rapaport/perry.positions.html

Rebelo, B. (2007). *SICAS-COL - Um Sistema Colaborativo para Aprendizagem Inicial da Programação.* Tese de Mestrado, Departamento de Engenharia Informática, Faculdade de Ciências e Tecnologia da Universidade de Coimbra, Coimbra, Portugal.

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

Reek, M. M. (1995). A top-down approach to teaching programming. *SIGCSE Bulletin, 27* (1), 6-9.

Reeves, M. F. (1990). An Application of Bloom's Taxonomy to the Teaching of Business Ethics. *Journal of Business Ethics, 9* (7), 609-616.

Reigeluth, C. M., & Stein, F. S. (1983). The elaboration theory of instruction. In C. M. Reigeluth (Ed.), *Instructional-Design Theories and Models: An Overview of their Current Status* (pp. 338-381). Lawrence Erlbaum Associates. Hillsdale, NJ, USA.

Reigeluth, C. M., Merrill, M. D., & Bunderson, C. V. (1994). The Structure of Subject Matter Content and its Instructional Design Implications. In M. D. Merril, & D. G. Twitchell (Ed.), *Instructional design theory* (pp. 59-77). Educational Technology Publications. Englewood Cliffs, NJ, USA.

Reigeluth, C. M., Merrill, M. D., Wilson, B. G., & Spiller, R. T. (1994). The Elaboration Theory of Instruction: A Model for Sequencing and Synthesizing Instruction. In M. D. Merril, & D. G. Twitchell (Ed.), *Instructional design theory* (pp. 79-102). Educational Technology Publications. Englewood Cliffs, NJ, USA.

Reynolds, C., & Fox, C. (1996). Requirements for a computer science curriculum emphasizing information technology: subject area curriculum issues. *SIGCSE Bulletin, 28* (1), 247-251.

Rich, C., & Waters, R. C. (1988). The Programmer's Apprentice: A Research Overview. *Computer, 21* (11), 10-25.

Riding, R., & Cheema, I. (1991). Cognitive Styles - An Overview and Integration. *Educational Psychology: An International Journal of Experimental Educational Psychology, 11* (3-4), 193-215.

Rist, R. S. (1989). Schema creation in programming. *Cognitive Science, 13* (3), 389-414.

Roberts, E. (2004). Resources to support the use of Java in introductory computer science. *SIGCSE Bulletin, 36* (1), 233-234.

Roberts, T. S. (2006). The use of multiple choice tests for formative and summative assessment. *ACE'06: Proceedings of the 8th Austalian Conference on Computing Education* (pp. 175-180). Australian Computer Society, Inc.. Hobart, Australia.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education, 13* (2), 137-172.

Roman, G.-C., & Cox, K. C. (1989). A Declarative Approach to Visualizing Concurrent Computations. *Computer, 22* (10), 25-36.

Roman, G.-C., Cox, K. C., Wilcox, C. D., & Plun, J. Y. (1992). Pavane: a system for declarative visualization of concurrent computations. *Journal of Visual Languages & Computing, 3* (2), 161-193.

Rosa, P. R., Moreira, M. A., & Buchweitz, B. (1992). Alunos bons solucionadores de problemas: caracterização a partir de um questionário para análise de entrevistas. *Revista Brasileira de Ensino de Física, 14* (2), 94-100.

Rosário, P. (1997). Aprendizagem auto-regulada. *Actas do 1º Congresso Luso-Espanhol de Psicologia da Educação* (pp. 405-414). Associação dos Psicólogos Portugueses. Coimbra, Portugal.

Rosário, P. (1999). As abordagens dos alunos ao estudo: Diferentes modelos e suas interrelações. *Psicologia: Teoria, Investigação e Prática, 4* (1), 43-61.

Rosário, P., Núñez, J., & González-Pienda, J. (2006). *Cartas do Gervásio ao seu Umbigo. Comprometer–se com o Estudar na Universidade.* Almedina. Coimbra, Portugal.

Rosati, P. A. (1995). Engineering Student Responses to an Index of Learning Styles. *Proceedings of ASEE Annual Conference*, (pp. 739-741). Anaheim, CA, USA.

Rosati, P. A. (1996). Comparisons of learning preferences in an engineering program. *FIE'96: Proceedings of the 26th ASEE/IEEE Annual Frontiers in Education Conference. 3*, pp. 1441-1444. IEEE Computer Society. Salt Lake City, UT, USA.

Rosati, P. A. (1999). Specific differences and similarities in the learning preferences of engineering students. *FIE'99: Proceedings of the 29th ASEE/IEEE Annual Frontiers in Education Conference. 2*, pp. 12c1.17-12c1.22. IEEE Computer Society. San Juan, Puerto Rico.

Rößling, G., & Naps, T. L. (2002). A testbed for pedagogical requirements in algorithm visualizations. *SIGCSE Bulletin, 34* (3), 96-100.

Rößling, G., Naps, T. L., Hall, M. S., Karavirta, V., Kerren, A., Leska, C., et al. (2006). Merging interactive visualizations with hypertextbooks and course management. *SIGCSE Bulletin, 38* (4), 166-181.

Roumani, H. (2006). Practice what you preach: full separation of concerns in CS1/CS2. *SIGCSE Bulletin, 38* (1), 491-494.

Roy, P., & St-Denis, R. (1976). Linear flowchart generator for a structured language. *SIGPLAN Notices, 11* (11), 58-64.

Rutz, E., Elkins, V., Rafter, C., Houshmand, A., & Eckart, R. (2000). Evaluation of Learning Styles and Instructional Technologies. *Proceedings of the 2000 ASEE Annual Conference & Exposition. Session 2793.* ASEE. St. Louis, MO, USA.

Sajaniemi, J. (2002). Visualizing roles of variables to novice programmers. In J. Kuljis, L. Baldwin, & R. Scoble (Ed.), *PPIG'2002: Proceedings of the 14th Workshop of the Psychology of Programming Interest Group*, (pp. 111-127). London, UK.

Sajaniemi, J., & Kuittinen, M. (2003). Program animation based on the roles of variables. *SoftVis'03: Proceedings of the 2003 ACM Symposium on Software Visualization* (pp. 7-16). ACM. San Diego, CA, USA.

Sanders, I., & Mueller, C. (2000). A fundamentals-based curriculum for first year computer science. *SIGCSE Bulletin, 32* (1), 227-231.

Santucci, U. (2000). *Problem setting*. Obtido em 1 de Abril de 2010, de http://www.problemsetting.it/

Satratzemi, M., Dagdilelis, V., & Evagelidis, G. (2001). A system for program visualization and problem-solving path assessment of novice programmers. *SIGCSE Bulletin, 33* (3), 137-140.

Scanlan, D. A. (1989). Structured Flowcharts Outperform Pseudocode: An Experimental Comparison. *IEEE Software, 6* (5), 28-36.

Schaub, S. (2000). Teaching Java with Graphics in CS1. *SIGCSE Bulletin, 32* (2), 71-73.

Scholtz, J., & Wiedenbeck, S. (1992). The role of planning in learning a new programming language. *International Journal of Man-Machine Studies, 37* (2), 191-214.

Schunk, D. H. (2000). *Learning theories: An educational perspective* (3rd ed.). Merril/Prentice-Hall, Inc.. Upper Saddle River, NJ, USA.

Scott, T. (2003). Bloom's taxonomy applied to testing in computer science classes. *Journal of Computing Sciences in Colleges, 19* (1), 267-274.

Seamster, T. L., Redding, R. E., & Kaempf, G. F. (1997). *Applied Cognitive Task Analysis in Aviation.* Ashgate Publishing. Aldershot, Hampshire, England.

Seamster, T. L., Redding, R. E., & Kaempf, G. L. (2000). A Skill-Based Cognitive Task Analysis Framework. In J. M. Schraagen, S. F. Chipman, & V. L. Shalin (Edits.), *Cognitive task analysis* (pp. 135-146). Lawrence Erlbaum Associates, Inc.. Mahwah, NJ, USA.

Seery, N., Gaughran, W. F., & Waldmann, T. (2003). Multi-Modal Learning in Engineering Education. *Proceedings of the 2003 ASEE Conference and Exposition. (CD-ROM).* ASEE. Nashville, TN, USA.

Selker, T. (1994). COACH: a teaching agent that learns. *Communications of the ACM, 37* (7), 92-99.

Sellman, R. (1992). Gravitas: An Object-Oriented Discovery Learning Environment for Newtonian Gravitation. In J. Gornostaev (Ed.), *EWHCI'92: Proceedings of the East-West International Conference on Human-Computer Interaction*, (pp. 31-41). St. Petersburg, Russia.

Shackelford, R. L., & Badre, A. N. (1993). Why can't smart students solve simple programming problems? *International Journal of Man-Machine Studies, 38* (6), 985-997.

Sheil, B. A. (1981). The Psychological Study of Programming. *ACM Computing Surveys (CSUR), 13* (1), 101-120.

Shiffrin, R. M., & Schneider, W. (1977). Controlled and Automatic Human Information Processing: II. Perceptual Learning, Automatic Attending, and a General Theory. *Psychological Review, 84* (2), 127-190.

Shu, N. C. (1988). *Visual programming.* Van Nostrand Reinhold Co.. New York, USA.

Simon, D. P., & Simon, H. A. (1978). Individual differences in solving physics problems. In R. Siegler (Ed.), *Children's thinking: What develops?* (pp. 325-348). Lawrence Erlbaum Associates, Inc.. Hillsdale, NJ, USA.

Simon, Fincher, S., Robins, A., Baker, B., Box, I., Cutts, Q., et al. (2006). Predictors of success in a first programming course. *ACE'06: Proceedings of the 8th Austalian Conference on Computing Education* (pp. 189-196). Australian Computer Society, Inc.. Hobart, Australia.

Simon, H. A. (1973). The structure of ill structured problems. *Artificial Intelligence, 4* (3-4), 181-201.

Simons, B., Frailey, D. J., Turner, A. J., Zweben, S. H., & Denning, P. J. (1991). An ACM response: The Scope and Directions of Computer Science. *Communications of the ACM, 34* (10), 121-131.

Simpson, E. J. (1966). The classification of educational objectives: Psychomotor domain. *Illinois Teacher of Home Economics, 10* (4), 121-126.

Slack, S. J., & Stewart, J. (1990). High school students' problem-solving performance on realistic genetics problems. *Journal of Research in Science Teaching, 27* (1), 55-67.

Sloan, E. D. (1998). Personality and Teaching/Learning Engineering. *Proceedings of the 1998 ASEE Annual Conference & Exposition. Session 2230.* ASEE. Seattle, WA, USA.

Sloane, K. D., & Linn, M. C. (1988). Instructional Conditions in Pascal Programming Classes. In R. E. Mayer (Ed.), *Teaching and Learning Computer Programming: multiple research perspectives* (pp. 207-235). Lawrence Erlbaum Associates, Inc.. Hillsdale, NJ, USA.

Smith III, J. P., diSessa, A. A., & Roschelle, J. (1993). Misconceptions Reconceived: A Constructivist Analysis of Knowledge in Transition. *Journal of the Learning Sciences, 3* (2), 115-163.

Smith, D. C., Cypher, A., & Spohrer, J. (1994). KidSim: programming agents without a programming language. *Communications of the ACM, 37* (7), 54-67.

Smith, M. U. (1988). Successful and unsuccessful problem solving in classical genetic pedigrees. *Journal of Research in Science Teaching, 25* (6), 411-433.

Smith, M. U., & Good, R. (1984). Problem solving and classical genetics: Successful versus unsuccessful performance. *Journal of Research in Science Teaching, 21* (9), 895-912.

Soloman, B. A., & Felder, R. M. (s.d.). *Index of Learning Styles Questionnaire*. Obtido em 1 de Abril de 2010, de http://www.engr.ncsu.edu/learningstyles/ilsweb.html

Soloway, E. (1986). Learning to program = learning to construct mechanisms and explanations. *Communications of the ACM, 29* (9), 850-858.

Soloway, E., & Spohrer, J. C. (1988). *Studying the Novice Programmer.* Lawrence Erlbaum Associates, Inc.. Hillsdale, NJ, USA.

Soloway, E., Bonar, J., & Ehrlich, K. (1983). Cognitive strategies and looping constructs: an empirical study. *Communications of the ACM, 26* (11), 853-860.

Soloway, E., Ehrlich, K., Bonar, J., & Greenspan, J. (1982). What do novices know about programming? (A. Badre, & B. Schneiderman, Edits.) *Directions in Human-Computer Interaction*, 87-122.

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.

Sosnovsky, S., Shcherbinina, O., & Brusilovsky, P. (2003). Web-based Parameterized Questions as a Tool for Learning. In A. Rossett (Ed.), *Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2003* (pp. 309-316). AACE. Phoenix, AZ, USA.

Spohrer, J. C., & Soloway, E. (1986). Novice mistakes: are the folk wisdoms correct? *Communications of the ACM, 29* (7), 624-632.

Spohrer, J. C., Soloway, E., & Pope, E. (1985). A goal/plan analysis of buggy pascal programs. *Human-Computer Interaction, 1* (2), 163-207.

Stamouli, I., & Huggard, M. (2006). Object oriented programming and program correctness: the students' perspective. *ICER '06: Proceedings of the Second International Workshop on Computing Education Research* (pp. 109-118). ACM. Canterbury, UK.

Stasko, J. T. (1990). Tango: A Framework and System for Algorithm Animation. *Computer, 23* (9), 27-39.

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

Stasko, J. T., & Kraemer, E. (1993). A methodology for building application-specific visualizations of parallel programs. *Journal of Parallel and Distributed Computing, 18* (2), 258-264.

Stasko, J. T., & McCrickard, D. S. (1995). Real Clock Time Animation Support for Developing Software Visualizations. *Australian Computer Journal, 27* (3), 118-128.

Stasko, J. T., & Patterson, C. (1992). Understanding and characterizing software visualization systems. *Proceedings of the IEEE 1992 Workshop on Visual Languages* (pp. 3-10). IEEE Computer Society Press. Seattle, WA, USA.

Sternberg, R. J., & Davidson, J. E. (1989). A Four-Prong Model for Intellectual-Skills Development. *Journal of Research and Development in Education, 22* (3), 22-28.

Stice, J. E. (1987). Using Kolb's Learning Cycle to Improve Student Learning. *Engineering Education, 77* (5), 291-296.

Stiggins, R. J. (2005). *Student-Involved Assessment for Learning.* Pearson/Merrill Prentice Hall. Upper Saddle River, NJ, USA.

Storey, M.-A., Sanseverino, M., German, D., Damian, D., Damian, A., Michaud, J., et al. (2003). Adopting GILD: An Integrated Learning and Development Environment for Programming. *ACSE 2003: 3rd International Workshop on Adoption-Centric Software Engineering, ICSE 2003 the 25th Internatioonal Conference on Software Engineering.* Portland,OR, USA.

Svec, S. (2005). Taxonomy for Teaching: A System for Teaching Objectives, Learning Activities and Assessment Tasks (Revision of Bloom's Taxonomy of the Cognitive Domain). *Pedagogicka revue (in Slovak), 57*, 453-476.

Swedish Ministry of Education and Research. (2003). *Higher Education Ordinance.* Obtido em 1 de Abril de 2010, de http://www.sweden.gov.se/sb/d/574/a/21541

Swenson, E. J. (1949). Organization and Generalization as Factors in Learning, Transfer, and Retroactive Inhibition. In E. J. Swenson, G. L. Anderson, & C. L. Stacey (Edits.), *Learning Theory in School Situations* (pp. 9-39). University of Minnesota Press. Minneapolis, MN, USA.

Taba, H. (1966). *Teaching Strategies and Cognitive Functioning in Elementary School Children.* Cooperative Research Project No. 2404, San Francisco State College, San Francisco, CA, USA.

Tavares, J., Almeida, L. S., Vasconcelos, R. M., & Bessa, J. (2004). *Inventário de Atitudes e Comportamentos Habituais de Estudo - IACHE.* Universidade de Aveiro/Universidade do Minho, Portugal.

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

Thompson, E. (2004). Does the sum of the parts equal the whole? In S. Mann, & T. Clear (Ed.), *Proceedings of the seventeenth annual conference of the National Advisory Committee on Computing Qualifications* (pp. 440-445). National Advisory Committee on Computing Qualifications. Christchurch, New Zealand.

Thompson, E. (2007). Holistic assessment criteria: applying SOLO to programming projects. *ACE'07: Proceedings of the Ninth Australasian Conference on Computing Education* (pp. 155-162). Australian Computer Society, Inc.. Ballarat, Victoria, Australia.

Thuné, M., & Eckerdal, A. (2009). Variation theory applied to students' conceptions of computer programming. *European Journal of Engineering Education, 34* (4), 339-347.

Tobón, R., & Perea, A. (1985). Problemas actuales en la enseñanza de la Física. *Revista de Enseñanza de la Física, 1* (1), 7-15.

Tomek, I. (1982). Josef, the Robot. *Computers & Education, 6* (3), 287-293.

Tomek, I., Muldner, T., & Khan, S. (1985). PMS - A program to make learning Pascal easier. *Computers & Education, 9* (4), 205-211.

Trætteberg, H., & Aalberg, T. (2006). JExercise: a specification-based and test-driven exercise support plugin for Eclipse. *Proceedings of the 2006 OOPSLA Workshop on Eclipse Technology eXchange*, (pp. 70-74). Portland, OR, USA.

Traynor, D., & Gibson, J. P. (2005). Synthesis and analysis of automatic assessment methods in CS1: generating intelligent MCQs. *SIGCSE Bulletin, 37* (1), 495-499.

Traynor, D., Bergin, S., & Gibson, J. P. (2006). Automated assessment in CS1. *ACE'06: Proceedings of the 8th Austalian Conference on Computing Education* (pp. 223-228). Australian Computer Society, Inc.. Hobart, Australia.

Truong, N., Bancroft, P., & Roe, P. (2003). A web based environment for learning to program. *ACSC'03: Proceedings of the 26th Australasian Computer Science Conference* (pp. 255-264). Australian Computer Society, Inc.. Adelaide, Australia.

Tucker, A. B. (Ed.). (1991). Computing Curricula 1991. *Communications of the ACM, 34* (6), 68-84.

Tucker, A. B. (1996). Strategic directions in computer science education. *ACM Computing Surveys (CSUR), 28* (4), 836-845.

Tyerman, S. P., Woods, P. J., & Warren, J. R. (1996). Loop Tutor and Hypertutor: experiences with adaptive tutoring systems. *Proceedings of the 1996 Australian and New Zealand Conference on Intelligent Information Systems*, (pp. 60-63). Adelaide, Australia.

Tyler, R. W. (1949). *Basic Principles of Curriculum and Instruction.* The University of Chicago Press. Chicago, IL, USA.

Ueno, H. (1989). INTELLITUTOR: a knowledge based intelligent programming environment for novice programmers. *COMPCON Spring'89: Proceedings of the Thirty-Fourth IEEE Computer Society International Conference: Intellectual Leverage, Digest of Papers.*, (pp. 390-395). San Francisco, CA, USA.

University of Illinois. (s.d.). *Sample Test Questions - Six Levels of Learning*. Obtido em 1 de Abril de 2010, de Illinois Online Network: Educational Resources: http://www.ion.illinois.edu/resources/tutorials/assessment/bloomtest.asp

University of Kent & Sun Microsystems. (2006). *The NetBeans IDE BlueJ Plugin*. Obtido em 1 de Abril de 2010, de NetBeans: http://edu.netbeans.org/bluej/

Varanda, M. J., & Henriques, P. R. (1999). Animação de Algoritmos tornada Sistemática. *Actas do 1º Workshop de Computação Gráfica, Multimédia e Ensino*, (pp. 7-15). Leiria, Portugal.

Ventura, P. R. (2005). Identifying predictors of success for an objects-first CS1. *Computer Science Education, 15* (3), 223-243.

Verco, K. L., & Wise, M. J. (1996). Plagiarism à la mode: A comparison of automated systems for detecting suspected plagiarism. *The Computer Journal, 39* (9), 741-750.

Visser, W., & Hoc, J.-M. (1990). Expert software design strategies. In J.-M. Hoc, T. R. Green, R. Samurçay, & D. J. Gilmore (Eds.), *Psychology of programming* (pp. 235-250). Academic Press. London, UK.

Vygotsky, L. S. (1978). *Mind in Society : The Development of Higher Psychological Processes.* (M. Cole, Ed.) Harvard University Press. Cambridge, MA, USA.

Waldheim, G. P. (1987). Understanding How Students Understand. *Journal of Engineering Education, 77* (5), 306-308.

Wallingford, E. (2000). Using patterns in the CS curriculum. *Journal of Computing Sciences in Colleges, 15* (5), 235-237.

Wang, Q., Kessler, G. D., Blank, G. D., & Pottenger, W. M. (2002). Demonstration tools for Collaborative E-Learning. *CSCW'2002: Proceedings Companion of the 2002 ACM Conference on Computer Supported Collaborative Work.* New Orleans, LA, USA.

Watkins, S. E., Huggans, M. A., & Nystrom, H. E. (2003). Learning Styles as a Design Parameter for Asynchronous Web-based Learning Modules. *Proceedings of the 2003 ASEE Annual Conference & Exposition. Session 1658.* ASEE. Nashville, TN, USA.

Weber, G. (1996). Individual selection of examples in an intelligent learning environment. *Journal of Artificial Intelligence in Education, 7* (1), 3-31.

Weber, G., & Brusilovsky, P. (2001). ELM-ART: An Adaptive Versatile System for Web-based Instruction. *International Journal of Artificial Intelligence in Education, 12* (4), 351-384.

Wegner, P., & Doyle, J. (1996). Editorial: Strategic directions in computing research. *ACM Computing Surveys (CSUR), 28* (4), 565-574.

Whale, G. (1994). DRUIDS: Tools for Understanding Data Structures and Algorithms. *Proceedings of the 1994 IEEE First International Conference on Multi-Media Engineering Education*, (pp. 403-407). Melbourne, VIC, Australia.

Whalley, J. L., Lister, R., Thompson, E., Clear, T., Robbins, P., Kumar, P. K., et al. (2006). An Australasian study of reading and comprehension skills in novice programmers, using the Bloom and SOLO taxonomies. *ACE'06: Proceedings of the 8th Austalian Conference on Computing Education* (pp. 243-252). Australian Computer Society, Inc.. Hobart, Australia.

Wiedenbeck, S. (1986). Processes in Computer Program Comprehension. In E. Soloway, & S. Iyengar (Edits.), *Empirical Studies of Programmers* (pp. 48-57). Ablex Publishing Corp.. Norwood, NJ, USA.

Wiedenbeck, S., Fix, V., & Scholtz, J. (1993). Characteristics of the mental representations of novice and expert programmers: an empirical study. *International Journal of Man-Machine Studies, 39* (5), 793-812.

Wilkerson, M., Griswold, W. G., & Simon, B. (2005). Ubiquitous presenter: increasing student access and control in a digital lecturing environment. *SIGCSE Bulletin, 37* (1), 116-120.

Williams, L. A., & Kessler, R. R. (2001). Experiments with Industry's "Pair-Programming" Model in the Computer Science Classroom. *Computer Science Education, 11* (1), 7-20.

Williams, L. A., & Tomayko, J. (2002). Agile Software Development. *Computer Science Education, 12* (3), 167-168.

Wilson, B. C., & Shrock, S. (2001). Contributing to success in an introductory computer science course: a study of twelve factors. *SIGCSE Bulletin, 33* (1), 184-188.

Winslow, L. E. (1996). Programming pedagogy—a psychological overview. *SIGCSE Bulletin, 28* (3), 17-22.

Wirth, N. (1971). Program Development by Stepwise Refinement. *Communications of the ACM, 14* (4), 221-227.

Woodford, K., & Bancroft, P. (2005). Multiple choice questions not considered harmful. *ACE'05: Proceedings of the 7th Australasian Conference on Computing Education* (pp. 109-116). Australian Computer Society, Inc.. Newcastle, New South Wales, Australia.

Woolfolk, A. E., Winne, P. H., & Perry, N. E. (2006). *Educational Psychology* (3rd Canadian ed.). Allyn & Bacon. Scarborough, Ontario, Canada.

Zimmerman, B. J. (1989). Models of self-regulated learning and academic achievement. In B. J. Zimmerman, & D. H. Schunk (Eds.), *Self-Regulated Learning and Academic Achievement: Theory, Research and Practice* (pp. 1-25). Springer-Verlag. New York, USA.

Zimmerman, B. J. (1990). Self-Regulated Learning and Academic Achievement: An Overview. *Educational Psychologist, 25* (1), 3-17.

Zualkernan, I. A., Allert, J., & Qadah, G. Z. (2006). Learning Styles of Computer Programming Students: A Middle Eastern and American Comparison. *IEEE Transactions on Education, 49* (4), 443-450.

Zywno, M. S. (2002). Instructional Technology, Learning Styles and Academic Achievement. *ASEE'2002: Proceedings of the 2002 American Society for Engineering Education Annual Conference & Exposition, Session 2422.* Montréal, Quebec, Canada.

Zywno, M. S. (2003). A Contribution of Validation of Score Meaning for Felder-Soloman's Index of Learning Styles. *Proceedings of the 2003 American Society for Engineering Education Annual Conference & Exposition. Session 2351.* ASEE. Nashville, TN, USA.

ANEXOS

Anexo A

Estudo A

Anexo A.1 – Teste de Diagnóstico de Resolução de Problemas

1. Qual é o número par maior que 20 e menor que 30 cuja soma dos seus algarismos é 8?

2. Uma pessoa encontra-se no degrau do meio de uma escada. Sobe 5 degraus, desce 7, volta a subir 4 e depois mais 9 para chegar ao último. Quantos degraus tem a escada? Explique o seu raciocínio.

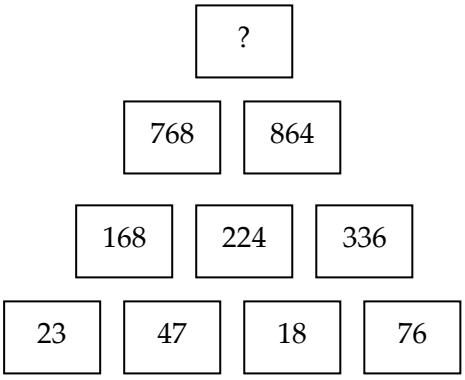
3. Numa prisão havia 3 presos. O director da prisão resolve conceder a liberdade a um deles, e propõe o seguinte:

   - Tenho aqui 5 bonés: 3 azuis e 2 amarelos. Inicialmente, quero que vocês fiquem os 3 alinhados em fila – desta forma o que está atrás vê os dois da frente, o do meio apenas vê o que se encontra à sua frente e o da frente não vê nenhum dos outros dois. Em seguida, vou colocar um boné, aleatoriamente, na cabeça de cada um. Sem se virarem, aquele que adivinhar a cor do seu boné será libertado!
   Após serem colocados os bonés, os dois de trás até riram do coitado que ficou na frente, pois ele não conseguia ver o boné de ninguém, e consequentemente não teria a mínima hipótese de adivinhar! Em seguida, foi feita a pergunta ao último da fila:
   - Qual a cor do seu boné?
   - Embora esteja a ver os bonés dos meus 2 companheiros, a minha resposta infelizmente é: Não sei!
   Foi feita então a pergunta ao do meio:
   - Qual a cor do seu boné?
   - Não sei!!!
   Foi feita então a pergunta ao da frente, que não estava a ver nada:
   - Qual a cor do seu boné?

- EU SEI!!!!! E serei libertado!

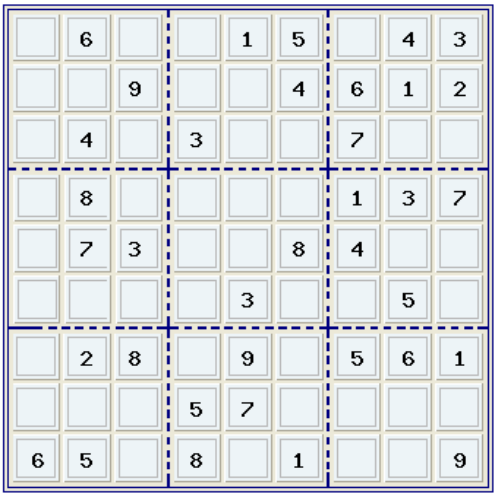
Qual a cor do boné do preso que será libertado? Explique o seu raciocínio.

4. Os números nos tijolos da base da figura seguinte são aleatórios. Contudo, os números de cada um dos tijolos do meio são criados – utilizando uma regra simples – a partir dos dois números dos tijolos imediatamente abaixo deles. Descubra a regra, de modo a encontrar o número do tijolo acima.



5. Dados os seguintes números 1, 2, 3, 4, 8, 11, 12, 20, 32, 44, 66, 70, 88 indique quais são os divisores de 22.

6. Preencha o seguinte tabuleiro, sabendo que o objectivo consiste em preencher o quadrado de 9x9, de tal forma que cada linha, coluna e caixa interior contenha os números de 1 a 9, sem se repetirem.

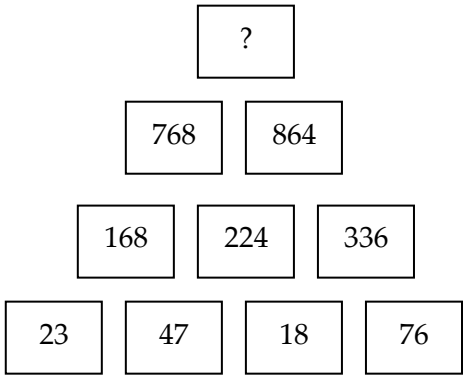


Anexo A.2 – Teste de Diagnóstico de Programação

1. Desenvolva um programa que peça dois valores que representam o comprimento de dois lados adjacentes de um polígono de 4 lados e o valor em graus do ângulo formado por esses dois lados. Sabendo que o polígono tem dois lados iguais 2 a 2 e ângulos também iguais 2 a 2, indique de que tipo de polígono se trata. O programa deve exibir os nomes dos possíveis polígonos.

2. Desenvolva um programa que descubra quais os números pares que satisfazem as seguintes condições: maior que determinado número (***menor***), menor que outro número (***maior***), não é o número ***numero2*** e a soma dos seus algarismos é igual a ***soma***.

3. Desenvolva um programa que peça ao utilizador um conjunto de números e que, a partir desses números, gere uma pirâmide de números, de tal forma que cada número acima dos dois abaixo seja obtido pela multiplicação de cada um dos dígitos que constituem cada um dos números que estão por baixo. Por exemplo, se o utilizador digitar os números 23, 47, 18 e 76, será obtida a seguinte pirâmide:



4. Dois números dizem-se amigos se a soma dos divisores de qualquer deles, incluindo a unidade e excluindo o próprio número, for igual ao outro número. Desenvolva um programa que permita verificar se dois números m e n são números amigos. Exemplo: 220 e 284 são números amigos.

   Divisores de 220: 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110; Soma: 284
   Divisores de 284: 1, 2, 4, 71, 142; Soma: 220

Anexo A.3 – Teste de Treino Nº1

1. Há três gémeos idênticos. O mais velho é o João, que diz sempre a verdade. O do meio é o António e diz sempre mentiras. O mais novo é o Joaquim e ora diz a verdade ora mente. Um dia o Tiago foi visitá-los e não conseguia distinguir quem era quem. Por isso fez uma pergunta a cada um deles. Perguntou ao que estava sentado à esquerda:

   - Quem está sentado ao teu lado?
   - É o João - foi a resposta.

   Perguntou ao que estava sentado ao meio:
   - Como te chamas?
   - Chamo-me Joaquim.

   Perguntou depois ao que estava sentado à direita:
   - Quem está sentado ao meio?
   - É o António - respondeu ele.

   O Tiago ficou completamente baralhado. Basicamente ele fez as mesmas três perguntas mas obteve respostas diferentes. Consegue ajudar o Tiago a descobrir quem é quem? Explique o seu raciocínio.

2. Apresente, passo-a-passo, o resultado da divisão formal de 12345 por 10, indicando em cada passo os seguintes elementos (dividendo (D), divisor (d), quociente (q) e resto (r)), através da seguinte representação:

   D | d
   r q

3. Dado um número qualquer, descreva o procedimento para obter a soma dos dígitos que o compõem, indicando adicionalmente os seguintes elementos:

   a) Dados de entrada;

   b) Resultados;

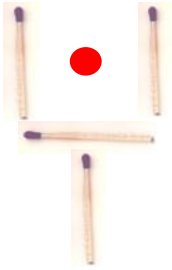
   c) Variáveis necessárias e valores de iniciação (caso necessário);

   d) Descreva o processo geral, de forma textual, incluindo os elementos pedidos nas alíneas anteriores;

   e) Represente graficamente cada um dos passos individuais de todo o processo, indicando, em cada um desses passos, os valores assumidos por cada um dos elementos pedidos nas alíneas anteriores.

Anexo A.4 – Teste de Treino Nº2

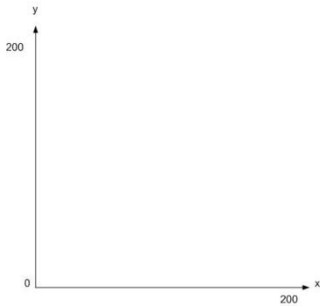
1. Suponha que existem 4 fósforos dispostos de acordo com o mostrado na figura seguinte. Descreva como poderia mover dois e apenas dois fósforos para novas posições, de maneira a reconstruir a mesma figura mas de forma a que o objecto existente no seu interior passe a situar-se no exterior.



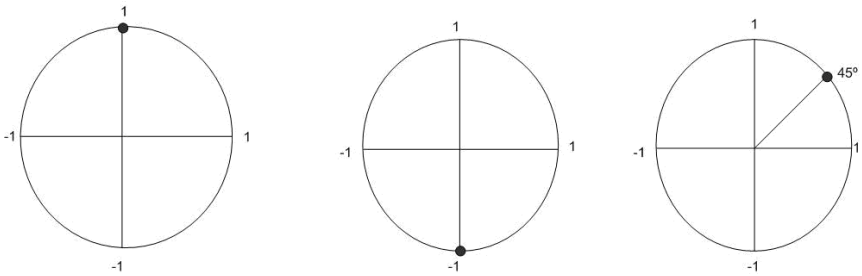
2. Suponha que tem um conjunto de 4 caixas quadradas de lados L1, L2, L3 e L4 e que L1<L2<L3<L4. Sabendo que L1 está dentro de L3 e L2 dentro de L4, descreva o procedimento para colocá-las todas umas dentro das outras.

3. Realize as seguintes tarefas:

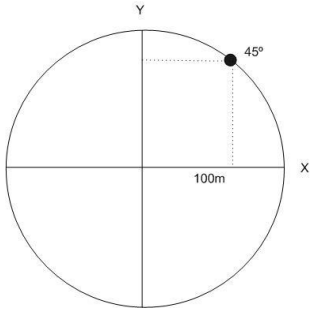
   a) Desenhe no plano cartesiano abaixo uma circunferência de raio 20 no ponto (50,50).



   b) Dadas as três circunferências abaixo, indique as coordenadas (x,y) dos pontos em destaque em cada uma delas.

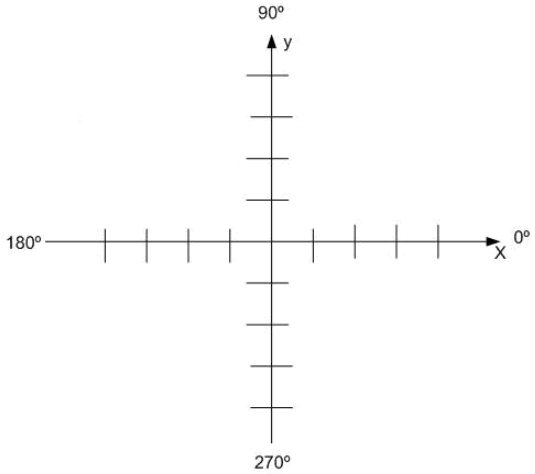


c) Suponhamos a seguinte circunferência com raio de 100 metros, indique a posição (em metros) do ponto destacado, no eixo x e y.

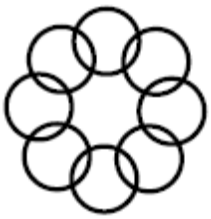


d) Desenhe quatro circunferências de raio 1 com centro nas seguintes coordenadas:

sen (0º), cos (0º) +1;
sen (90º) + 1, cos (90º);
sen (180º) , cos (180º) -1;
sen (270º) -1, cos (270º)



e) Descreva o procedimento para desenhar a figura seguinte, com centro no ponto (100, 100) e raio das circunferências 30.



Anexo A.5 – Teste de Treino Nº3

1. Numa determinada cidade a idade a partir da qual são autorizadas bebidas alcoólicas é de 21 anos. Cada uma das cartas abaixo fornece informação acerca de uma de 4 pessoas sentadas à mesa de um restaurante dessa cidade. Cada carta tem 2 lados. Um dos lados indica a idade da pessoa e o outro lado indica quando é que a pessoa em causa está ou não a beber cerveja. O gerente do restaurante faz a seguinte afirmação acerca da pessoa que está sentada à mesa:

   "Se a pessoa está a beber cerveja, então essa pessoa tem pelo menos 21 anos."
   Há suspeitas de que a declaração do gerente possa ser falsa. Que carta ou cartas necessitam de ser viradas para determinar quando é que essa declaração é falsa?

A beber cerveja	A beber coca-cola	20 anos	24 anos

   a) É necessário virar a carta com a inscrição "A beber cerveja" para verificar se a afirmação é falsa? Porquê?

   b) É necessário virar a carta com a inscrição "A beber coca-cola" para verificar se a afirmação é falsa? Porquê?

   c) É necessário virar a carta com a inscrição "20 anos" para verificar se a afirmação é falsa? Porquê?

   d) É necessário virar a carta com a inscrição "24 anos" para verificar se a afirmação é falsa? Porquê?

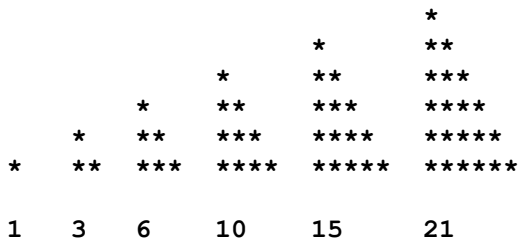
   Justifique as suas respostas em termos de proposições **E** e/ou **OU**.

2. O modo de entrega de notas de uma caixa Multibanco pode ser programada de várias formas. Uma delas consiste em atribuir ao utilizador o menor número de notas possíveis. Descreva o processo para determinar esse número. Assuma que a máquina dispõe de notas de 5, 10, 20 e 50 euros e que a quantia desejada pelo utilizador é um múltiplo de 5 euros.

3. Pitágoras concebeu os números triangulares, cada um dos quais corresponde à soma dos primeiros números naturais, por exemplo, 1=1; 3=1+2; 6=1+2+3; 10=1+2+3+4; 15=1+2+3+4+5; etc.

   a) Indique um processo para verificar se um determinado número é triangular.

b) Os números triangulares, podem constituir números figurados na medida em que formam figuras geométricas, neste caso triângulos como é exemplificado de seguida.



Indique um processo para gerar uma das 6 figuras geométricas acima de acordo com um número indicado pelo utilizador.

c) Altere a resolução anterior, de forma a gerar a figura abaixo, supondo que o triângulo tem uma altura n=6.

Anexo A.6 – Teste de Treino Nº4

1. Temos dez pilhas de moedas, a 1ª com uma moeda, a 2ª com duas moedas, e assim sucessivamente até à 10ª que possui 10 moedas. Uma das pilhas é formada somente por moedas falsas. Sabe-se que as moedas verdadeiras pesam dois gramas e que as moedas falsas pesam um grama a menos que as genuínas. Determine o menor número de pesagens necessárias para descobrir a pilha de moedas falsas, usando uma balança de um só prato.

2. Três amigos tiveram um almoço de negócios e concordaram em dividir a conta. O empregado apresentou-lhes uma conta de 30€ e cada um deles entregou-lhe uma nota de 10€. Porém, tendo posto o dinheiro na bandeja, o empregado apercebeu-se que lhes tinha levado dinheiro a mais. A conta deveria ter sido de 26€, pois havia um desconto especial no vinho, no qual ele não tinha reparado. Assim, tirou 4€ da bandeja a fim de lhos devolver. Contudo, não sabia como dividir equitativamente 4€ por três pessoas, pelo que meteu 1€ no bolso e deu 1€ a cada um dos amigos. Assim, cada amigo pagou 9€ e entre os três pagaram 27€. Mas o empregado tinha apenas 1€ no bolso, perfazendo um total de 28€. O que aconteceu então aos outros 2€?

3. Complete a grelha abaixo de modo a que em cada linha e em cada coluna estejam as cinco letras que aparecem na grelha. Cada um dos grupos de 5 quadrados delimitados pelas linhas a cheio deve também conter cada uma das referidas letras.

	A			B
E				
	D			C

Anexo A.7 – Teste Final de Programação

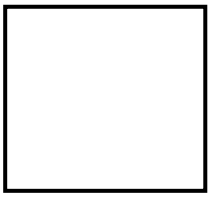
Enquadramento

Um **paralelogramo** é um quadrilátero (polígono de 4 lados) cujos lados opostos são iguais e paralelos. Consequentemente tem ângulos opostos iguais.

Classificações

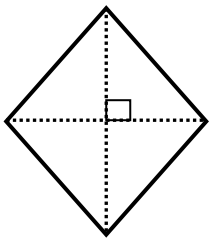
De interesse para este exercício considere os seguintes tipos de paralelogramos: o **quadrado**, o **rectângulo** e o **losango**.

Um **quadrado** é um paralelogramo que tem todos os lados iguais e os 4 ângulos iguais (90º).



Um **rectângulo** é um paralelogramo cujos lados formam ângulos rectos (90º) entre si. O comprimento dos seus lados é igual dois a dois.

Um **losango** é um paralelogramo cujos lados são de igual comprimento. As diagonais de um losango formam um ângulo de 90º.



1. Desenvolva um programa que peça dois valores que representam o comprimento de dois lados adjacentes de um polígono de 4 lados e o valor em graus do ângulo formado por esses dois lados. Sabendo que o polígono tem os lados iguais 2 a 2 e os ângulos também iguais 2 a 2, indique de que tipo de polígono se trata. O programa deve exibir os nomes dos possíveis polígonos.

2. Assumindo a existência das seguintes funções:

```
IniGraph(int width, int height, int nr_colours)
```

Inicia o sistema gráfico para funcionamento em ecrãs de dimensão (*width,heigth*) e com suporte para *nr_colours* cores em simultâneo. O canto superior

esquerdo corresponde às coordenadas (0,0) e o canto inferior direito às coordenadas (*width-1*,*height-1*).

```
DrawOval(int left, int top, int width, int height)
```

Que desenha uma oval dentro do rectângulo definido pelos parâmetros da função.

Diga qual o comportamento do seguinte programa:

```
void main(int argc, char **argv) {
    double PI= 3.14159265358979323846;
    int dimensao=50;
    int i, posx, posy;
    IniGraph(640,480,16);
    for(i=0;i<=360;i+=45){
        posx=(int)(cos(i*PI/180)*dimensao);
        posy=(int)(sin(i*PI/180)*dimensao);
        DrawOval((200+posx)-25,(200+posy)-25,50,50);
    }
}
```

Referindo:

a) O valor assumido por cada variável em cada iteração.

b) O resultado final do programa.

3. Dois números dizem-se amigos se a soma dos divisores de qualquer deles, incluindo a unidade e excluindo o próprio número, for igual ao outro número. Desenvolva um programa que permita verificar se dois números m e n são números amigos. Exemplo: 220 e 284 são números amigos.

   Divisores de 220: 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110; Soma: 284

   Divisores de 284: 1, 2, 4, 71, 142; Soma: 220

   Assuma que os dois números inteiros estão armazenados num ficheiro de texto, cujo nome é passado ao programa através da linha de comando. Este ficheiro contém uma única linha e os números estão separados por um ou mais espaços em branco.

Anexo A.8 – Normas Matemáticas

Serão descritas neste anexo, as normas que foram utilizadas no desenrolar das experiências, em função do contributo que pensamos poderem fornecer para a capacidade de programar.

Norma 1: A Matemática como Resolução de Problemas

Esta norma lida com conceitos matemáticos necessários à resolução de problemas e sua aplicação em situações fora da matemática, bem como a generalização de soluções e estratégias para novas situações. Assim, esta norma recomenda que a resolução de problemas deva ser um método de investigação e aplicação, de forma a que os alunos:

- investiguem e compreendam conceitos matemáticos;
- formulem problemas a partir de situações dentro e fora da matemática;
- desenvolvam e apliquem uma variedade de estratégias para resolver problemas, com ênfase em problemas não rotineiros e de solução não imediata;
- verifiquem e interpretem resultados relativos a um dado problema;
- generalizem soluções e estratégias para novas situações problemáticas;
- adquiram confiança na utilização da matemática de forma significativa.

Considerámos esta norma de crucial importância para o desenvolvimento da capacidade de resolução de problemas de programação, em especial pelas capacidades de abstracção associadas.

Norma 2: A Matemática como Comunicação

Esta norma refere a importância de incluir oportunidades de comunicação, de forma a que os alunos:

- criem modelos de situações através da expressão oral e escrita, usando objectos, desenhos e gráficos ou ainda por métodos algébricos;
- reflictam e clarifiquem o seu próprio pensamento acerca de ideias e situações matemáticas;
- desenvolvam a compreensão das ideias matemáticas, incluindo o papel das definições;

- utilizem as capacidades de ler, ouvir e ver para interpretar e avaliar ideias matemáticas;
- discutam ideias matemáticas e construam conjecturas e argumentos convincentes;
- apreciem o valor da notação matemática e o seu papel no desenvolvimento das ideias matemáticas.

Cremos na importância desta norma pois, um dos obstáculos com que os alunos se deparam na resolução de um problema de programação reside na tradução de uma solução textual para a linguagem matemática.

Norma 3: A Matemática como Raciocínio

Esta norma refere a importância da inclusão de assuntos de forma a que os alunos sejam capazes de:

- reconhecer e aplicar o raciocínio indutivo e dedutivo;
- compreender e aplicar processos de raciocínio, com especial atenção ao raciocínio espacial e ao raciocínio com proporções e gráficos;
- formular e avaliar conjecturas e argumentos matemáticos;
- validar o seu próprio pensamento;
- apreciar o uso e poder do raciocínio como parte da Matemática.

Esta norma é importante, em especial pela possibilidade de identificação de regularidades, característica fundamental do raciocínio indutivo, para além dos aspectos de compreensão e validação de diferentes tipos de raciocínios.

Norma 4: Conexões Matemáticas

Esta norma refere a importância da ênfase das conexões matemáticas de forma a permitir aos alunos:

- identificar a matemática como um todo;
- explorar problemas e descrever resultados utilizando modelos e representações gráficas, numéricas, físicas, algébricas e verbais;
- utilizar uma ideia matemática para aprofundar a sua compreensão acerca de outras ideias matemáticas;

- aplicar o pensamento matemático e a modelação para resolver problemas que surgem noutras disciplinas;
- valorizar o papel da matemática na nossa cultura e na nossa sociedade.

Cremos na importância desta norma, em especial na potencialidade de estabelecer conexões matemáticas com situações concretas do quotidiano, familiares aos alunos, para resolver problemas de programação.

Norma 5: Números e Relações entre Números

Esta norma refere a importância da inclusão do desenvolvimento continuado dos números e das relações entre eles, de modo a que os alunos:

- compreendam, representem e utilizem números numa variedade de formas equivalentes (inteiros, fracções, decimais, percentagens, potências, notação científica) em situações problemáticas do mundo real e da matemática;
- desenvolvam o sentido do número para números inteiros, fracções, decimais, inteiros relativos e números racionais;
- compreendam e apliquem razões, proporções, e percentagens numa variedade de situações;
- investiguem relações entre fracções, decimais e percentagens;
- representem relações numéricas em gráficos a uma e duas dimensões.

Cremos que esta norma apresenta alguma importância no sentido de evidenciar que existem várias formas de representar a mesma coisa, mas que existem situações/contextos que fazem com que certas representações sejam mais adequadas ou façam mais sentido do que outras.

Norma 6: Sistemas Numéricos e Teoria dos Números

Esta norma refere a importância da inclusão do estudo de sistemas numéricos e teoria dos números de forma a que os alunos:

- compreendam e apreciem a necessidade da existência de números para além dos números inteiros;
- desenvolvam e usem relações de ordem para os números inteiros, os números fraccionários, os decimais, os inteiros relativos e os números racionais;

- alarguem a sua compreensão das operações com números inteiros às fracções, aos decimais, aos inteiros relativos e aos números racionais;
- compreendam como as operações aritméticas básicas estão relacionadas umas com as outras;
- desenvolvam e apliquem conceitos da teoria dos números (por exemplo, números primos, divisores e múltiplos) a situações problemáticas do mundo real.

Cremos na importância desta norma, na medida em que oferece uma multiplicidade de oportunidades para explorações interessantes, com consequências positivas na capacidade de resolução de problemas, na compreensão e desenvolvimento de outros conceitos matemáticos (números abundantes, deficientes, perfeitos, triangulares, quadrados, cubos, capicuas, factoriais, números de Fibonacci, máximo e mínimo divisor comum de dois números) de grande utilidade para a aprendizagem da programação.

Norma 7: Cálculo e Estimação

Esta norma refere a importância do desenvolvimento dos conceitos subjacentes ao cálculo e à estimação em vários contextos, de forma a que os alunos:

- calculem com números inteiros, fracções, decimais, inteiros relativos e números racionais;
- desenvolvam, analisem, e expliquem procedimentos de cálculo e técnicas de estimação;
- desenvolvam, analisem e expliquem métodos para resolver proporções;
- seleccionem e usem um método apropriado entre o cálculo mental, o cálculo com papel e lápis, a utilização da calculadora e a utilização do computador;
- usem o cálculo, a estimação e as proporções para resolver problemas;
- usem a estimação para avaliar resultados.

Cremos na importância desta norma, não apenas, para resolver problemas, mas para avaliar resultados. Consideramos, que a avaliação e análise de resultados errados podem constituir oportunidades de aprendizagem importantíssimas se os alunos descreverem o que estão a fazer e verificarem as suas respostas, em especial em termos de programação onde surgem constantemente erros devido a efeitos colaterais não previstos.

Norma 8: Padrões e Funções

Esta norma refere a importância da inclusão de explorações de padrões e funções, de forma a que os alunos:

- descrevam, ampliem, analisem e criem uma larga diversidade de padrões;
- descrevam e representem relações com tabelas, gráficos e regras;
- analisem relações funcionais para explicar como uma mudança numa quantidade implica uma mudança numa outra;
- usem padrões e funções para representar e resolver problemas.

Cremos que um aspecto importante desta norma, com reflexos na programação, diz respeito à capacidade de resolver um problema real expressando a solução através de uma função, de grande importância para questões relacionadas com a capacidade de abstracção.

Norma 9: Álgebra

Esta norma refere a importância da inclusão de explorações de conceitos e processos algébricos de tal modo que os alunos:

- compreendam os conceitos de variável, expressão e equação;
- representem situações e padrões numéricos com tabelas, gráficos, regras verbais, e equações e explorem as relações entre essas representações;
- analisem tabelas e gráficos para identificar propriedades e relações;
- ganhem confiança na resolução de equações lineares usando métodos concretos, informais e formais;
- investiguem inequações e equações não lineares informalmente;
- apliquem métodos algébricos para resolver uma diversidade de problemas do mundo real e da matemática.

Cremos na importância desta norma pois, uma das suas características, com implicação na programação, reside na compreensão dos conceitos de variável, expressão e equação. Em termos de programação o conceito de variável é, por vezes, um dos primeiros problemas com que os alunos se defrontam.

Norma 12: Geometria

Esta norma refere a importância da inclusão do estudo da geometria a duas e três dimensões, em situações muito diversas, de forma a que os alunos:

- identifiquem, descrevam, comparem, e classifiquem figuras geométricas;
- visualizem e representem figuras geométricas, com atenção especial para o desenvolvimento do sentido espacial;
- explorem transformações de figuras geométricas;
- representem e resolvam problemas usando modelos geométricos;
- compreendam e apliquem propriedades geométricas e relações;
- desenvolvam uma apreciação da geometria como uma forma de descrever o mundo físico.

Cremos nas capacidades conferidas por esta norma para a capacidade de programação. Directamente, porque existem numerosos problemas de programação que implicam que os alunos saibam identificar, descrever, comparar e classificar figuras geométricas, bem como outros que só podem ser resolvidos por aplicação de modelos geométricos. Indirectamente, pelas potencialidades referentes às capacidades de abstracção desenvolvidas.

Da totalidade das normas estabelecidas no escalão seguido, as referentes a Estatística, Probabilidades e Medida não foram descritas porque não foram contempladas nos exercícios envolvidos nas experiências. Isto porque, apesar de considerarmos que as capacidades por elas conferidas são importantes para conseguir obter e discernir informação, considerou-se que as competências básicas exigidas aos alunos neste domínio são suficientes para alcançar os objectivos pretendidos.

Anexo B

Estudo B

Anexo B.1 – Teste de Diagnóstico

1. O modo de entrega de notas de uma caixa Multibanco pode ser programada de várias formas. Uma delas consiste em atribuir ao utilizador o menor número de notas possível. Descreva o processo de determinar esse número. Assuma que a máquina dispõe de notas de 5 €, 10 €, 20 € e 50 € e que a quantia desejada pelo utilizador é um múltiplo de 5 €.

2. Numa determinada cidade a idade a partir da qual são autorizadas bebidas alcoólicas é de 21 anos. Cada uma das cartas abaixo fornece informação acerca de cada uma de 4 pessoas sentadas à mesa de um restaurante dessa cidade. Cada carta tem 2 lados. Um dos lados indica a idade da pessoa e o outro lado indica quando é que a pessoa em causa está ou não a beber cerveja. O gerente do restaurante faz a seguinte afirmação acerca da pessoa que está sentada à mesa: **"Se a pessoa está a beber cerveja, então essa pessoa tem pelo menos 21 anos."** Há suspeitas de que a declaração do gerente possa ser falsa. No entanto a informação presente nas cartas é sempre verdadeira. Que carta ou cartas necessitam de ser viradas para determinar quando é que essa declaração é falsa? Justifique a sua resposta.

A beber cerveja	A beber coca-cola	20 anos	24 anos

Anexo B.2 – Questionário

Este questionário tem como objectivo a melhor compreensão do perfil dos novos alunos do Departamento de Engenharia Informática.

Nome: _____Nº _____

Idade: _____ anos

Área frequentada (10º-12º anos)?_____

Média final do secundário: _____ valores

Média de acesso ao curso: _____ valores

Classificações, na prova de acesso, de cada uma das seguintes disciplinas:

Matemática: ____; Física: ____; Geometria Descritiva: ____

Em que opção colocou o curso que está a frequentar? ____

Que cursos/áreas colocou nas 3 primeiras preferências:

a) _____

b) _____

c) _____

Numa situação ideal (Nota de candidatura de 20 valores) que cursos colocaria nas 3 preferências mais elevadas.

a) _____

b) _____

c) _____

Em que linguagens de programação é capaz de programar? Como classifica os seus conhecimentos?

a) ☐ Pascal (___básicos; ___ médios; ___avançados)

b) ☐ C (___básicos; ___ médios; ___avançados)

c) ☐ Java (___básicos; ___ médios; ___avançados)

d) ☐ Python (___básicos; ___ médios; ___avançados)

e) ☐ Outra _____ (___básicos; ___ médios; ___avançados)

Observações:_____

_____

_____

Qual das seguintes frases melhor descreve a sua atitude relativamente ao curso em que ingressou?

a) ☐ Eu quero obter bons resultados para minha própria satisfação.

b) ☐ Eu quero obter bons resultados para agradar aos meus pais, família e amigos.

c) ☐ Eu quero obter bons resultados para agradar aos meus professores.

d) ☐ Eu quero obter bons resultados para conseguir uma boa profissão.

e) ☐ O meu objectivo principal é passar.

Obrigado pela colaboração.

Anexo C

Estudo C

Anexo C.1 – Teste 1.1

1. Determine o m.m.c. entre 6 e 21.

2. Escreva um programa que calcule o m.m.c. entre dois números.

3. Imagine que pretende fazer um programa para calcular a soma de duas fracções.

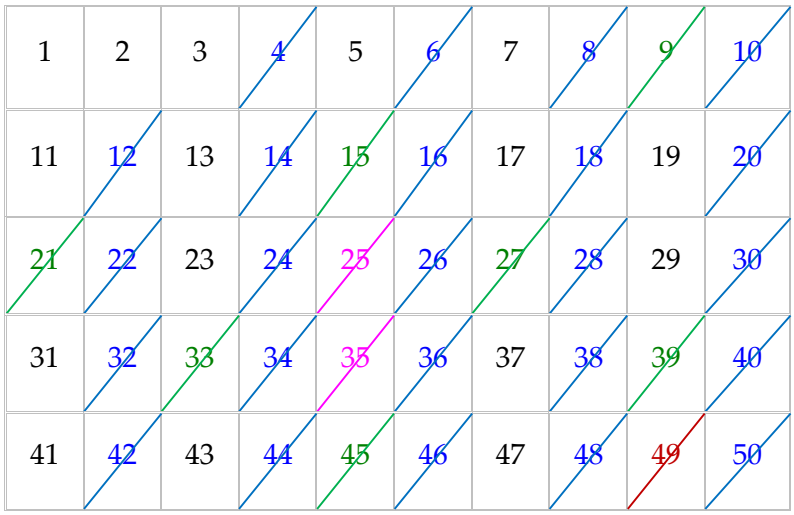
   a) Transforme o programa anterior numa função que receba dois números e calcule e devolva o m.m.c. entre eles.

   b) Faça o algoritmo do programa que calcula a soma de duas fracções, usando a função anterior. Devem ser introduzidos os 4 valores correspondentes às fracções N1/D1 e N2/D2 e mostrado o resultado da soma no formato N3/D3.

4. Do seguinte conjunto {2, 3, 4, 6, 7, 21, 37, 125} indique os que são números primos.

5. Escreva um programa que calcule e imprima todos os números primos existentes entre dois valores n1 e n2.

6. O Crivo de Eratóstenes é um método que permite obter uma tabela de números primos até um limite escolhido, através do seguinte procedimento: Escreve-se a sucessão natural dos números inteiros até ao número desejado. Suprime-se o número 1. O número 2 é o menor número primo. A partir do número que se lhe segue, o 3, eliminam-se todos os múltiplos de 2. O número 3, o primeiro que não foi eliminado, é primo. A partir do número que se lhe segue, o 5, eliminam-se todos os múltiplos de três. O número 5, não foi eliminado, é primo. A partir do número que se lhe segue, o 7, eliminam-se todos os múltiplos de cinco. O processo repete-se até terem sido percorridos todos os números da tabela. A figura seguinte ilustra este procedimento.



Escreva um programa que implemente o crivo de Eratóstenes até um número n.

Anexo C.2 – Teste 1.2

Conceito abordado neste teste: mínimo múltiplo comum (m.m.c.).

O m.m.c. de dois inteiros a e b é o menor inteiro positivo que é múltiplo simultaneamente de a e de b.

Por exemplo, sabendo que uma florista vende ramos de rosas a 3€ cada um, qual é o preço de 2, 3, 4 e 5 ramos? Resposta: 6€; 9€; 12€ e 15€, respectivamente.

Assuma agora que a florista vende, além de ramos de rosas a 3€, ramos de cravos a 6€ cada um. Suponha que o Sr. João queria pagar a mesma quantia pelos ramos de rosas e pelos ramos de cravos, mesmo não levando o mesmo número de ramos de cada um dos tipos. Indique duas possibilidades de preço para o Sr. João. Resposta: Calcular múltiplos de 6 e extrair dois comuns aos de 3. Por exemplo 6 e 12.

Nestas condições quanto teria de gastar no mínimo o Sr. João? Resposta: m.m.c (3,6)= 6€.

1. Identifique a resposta correcta, o m.m.c. entre 7 e 21 é:
   a) 3
   b) 147
   c) 21
2. Determine o m.m.c (12,20,24).
3. Dois amigos encontraram-se na cidade onde nasceram em Dezembro de 2007. Um deles regressa de 3 em 3 meses e o outro de 4 em 4 meses. Quando é que os amigos se voltaram a encontrar, na cidade considerada?
4. O que aconteceria se existisse um terceiro amigo que se encontrasse, na cidade, no mesmo dia dos outros dois e que retornasse à cidade de 5 em 5 meses? Quando é que os três se voltariam a encontrar?

Anexo C.3 – Teste 1.3

1. Dado o seguinte extracto de código

```
1:  var n1, n2, i, j :integer;
...
2:  n1:=3;
3:  n2:=10;
4:  for i:=n1 to n2 do
5:      if i mod 2 = 0
6:          then write(i);
```

   a) Identifique as variáveis.

   b) Qual o valor inicial de cada uma das variáveis? Indique o número da linha de código onde a variável é iniciada.

   c) Supondo i=57 qual o resultado da seguinte expressão `i:=i+1`?

   d) Supondo i=101 qual o resultado da seguinte expressão `i mod 2`?

   e) Existe algum ciclo? Se sim, indique qual a linha em que é iniciado e terminado?

   f) Existe alguma instrução de entrada e/ou saída de dados? Se sim, indique quais e em que linha(s).

   g) Explique o que faz a instrução `for i:=n1 to n2 do`.

2. Explique, de forma sintetizada, o que faz o pedaço de código acima.

3. Assuma que as linhas 2 e 3 são substituídas pelo seguinte código:

```
2:  n1:=10;
3:  n2:=3;
```

   Qual o resultado desta modificação no código acima?

4. Introduza as alterações necessárias no programa dado (em 1), considerando os dados da pergunta anterior, de forma a que o comportamento do programa seja o mesmo que o dado (em 1).

5. Explique a razão da condição `if i mod 2 = 0`?

6. Com base no programa dado, escreva um programa que imprima os múltiplos de n1 existentes entre n1 e n1*n2.

7. Evolua o programa anterior, de modo a imprimir os múltiplos comuns de n1 e n2, pertencentes ao intervalo entre n1 e n1*n2.

8. Altere o programa anterior de forma a que seja apenas mostrado o mínimo múltiplo comum entre n1 e n2.

9. Será necessário o ciclo percorrer todos os números intermédios entre n1 e n2? Que alterações sugere para optimizar o procedimento de cálculo do mínimo múltiplo comum entre n1 e n2.

Anexo C.4 – Teste 2.1

1. Será 3204 um múltiplo de 4? Justifique.

2. Qual é a decomposição em factores primos de 3780?

3. Observe as seguintes decomposições dos números A, B e C:

   $A = 2^3 \times 5 \times 7^2$ $\quad$ $B = 2^3 \times 5^2 \times 7^2$ $\quad$ $C = 2^3 \times 5^3 \times 7^2 \times 11$

   Sem efectuar cálculos indique, justificando, se:

   a) O número B é múltiplo de A;

   b) O número B é múltiplo de C;

   c) Por quanto temos de multiplicar A para obter C?

4. Considere os números 12, 18 e 90.

   a) Decomponha cada um dos números anteriores em factores primos.

   b) Identifique os factores primos comuns entre os números decompostos em a) e multiplique-os considerando os de maior expoente. Que número obteve?

   c) Obtenha o m.m.c. entre os números 12, 18 e 90.

5. Será possível obter o m.m.c. entre dois números à custa das suas factorizações? Dos métodos seus conhecidos, qual o que considera mais eficiente para a obtenção do m.m.c. entre números? Justifique.

Anexo C.5 – Teste 2.2

1. Dado o seguinte extracto de código

```
1:  PROGRAM XPTO(INPUT, OUTPUT);
2:  VAR d1, d2, maior, menor, i:INTEGER;
3:  BEGIN
4:      d1:=3;
5:      d2:=6;
6:      IF (d1>d2) THEN
7:          BEGIN
8:              maior:=d1;
9:              menor:=d2;
10:         END
11:     ELSE
12:         BEGIN
13:             maior:=d2;
14:             menor:=d1;
15:         END;
16:     FOR i:=maior TO maior*menor DO
17:         IF ((i mod maior=0) AND (i mod menor=0)) THEN
18:             break;
19:     WRITELN (i);
20: END.
```

   a) Identifique as variáveis.

   b) Qual o valor inicial de cada uma das variáveis? Indique o número da linha de código onde a variável é iniciada.

   c) Existe algum ciclo? Se sim, indique qual a linha em que é iniciado e terminado?

   d) Existe alguma instrução de entrada e/ou saída de dados? Se sim, diga quais e em que linha(s). Se não, indique a forma como são obtidos os valores das variáveis.

   e) Explique o que faz a instrução `FOR i:=maior TO maior*menor.`

   f) Explique, de forma sintetizada, o que faz e qual a utilidade do pedaço de código compreendido entre as linhas 6 e 15.

   g) Explique, de forma sintetizada, o que faz o pedaço de código compreendido entre as linhas 16 e 19.

   h) Explique o que faz a totalidade do código acima.

2. Escreva um programa que apresente no ecrã os divisores comuns entre dois números.

3. Dada a seguinte função cujo objectivo é determinar se um número é ou não primo, deduza qual a condição do `repeat…until`, completando-a através da escolha de uma das opções abaixo. Justifique a sua opção.

```
1:Function xpto (i:INTEGER):BOOLEAN;
2:  BEGIN
3:      i:=30;
4:      xpto:=TRUE;
5:      IF i>4 THEN
6:          BEGIN
7:              j:=2;
8:              REPEAT
9:                  IF (i mod j=0) THEN
10:                     xpto:=FALSE;
11:                 j:=j+1;
12:             UNTIL COMPLETAR
13:         END;
14: END
```

a) `UNTIL (xpto=false) or (j>=i).`

b) `UNTIL (xpto=false) or (j<=i).`

c) `UNTIL (xpto=true) or (j>=i).`

d) `UNTIL (xpto=true) or (j<=i).`

4. Sabendo que o Teorema da factorização única diz que todo o número maior que 1 pode ser escrito de um só modo, como um produto de números primos, escreva um programa que apresente no ecrã a decomposição de um número em factores primos. Por exemplo, supondo o número 84, no ecrã deverá surgir 2,2,3,7.

5. Suponha que "primo" é uma função que recebe um número e que devolve um valor (TRUE ou FALSE) em função do facto de o número que recebe ser primo ou não. Admita ainda que n1 é um número acerca do qual se quer obter a factorização em números primos, indique a utilidade/significado do seguinte extracto de código.

```
i:=2;
while (!primo(i) or (n1 mod i!=0))
    i:=i+1;
```

Anexo C.6 – Teste 2.3 – Conceitos matemáticos

Número primo

Um número primo p é um número inteiro p>1 que não tem divisores inteiros a não ser o número 1 e p. Se p não é um número primo diz-se que é um número composto.
Exemplo: Os únicos divisores do número 7 são o número 1 e o número 7, enquanto o número 24 tem por divisores os números 1, 2, 3, 4, 6, 8, 12 e 24, logo não é um número primo.

Algoritmo da Factorização em números primos

Este algoritmo pode ser aplicado a qualquer número natural p e permite-nos verificar se p é um número primo ou composto e, neste caso, quais são os seus factores primos. Para o implementar procede-se da seguinte forma: tenta-se, sucessivamente, dividir p por cada número primo n = 2, 3, 7, … até ao maior numero primo não superior a $\sqrt{p}$. Se o resto da divisão for sempre diferente de zero então p é um número primo. Se, digamos, p0, dividir p, p é composto e escreve-se p=p0p1, com p1<p, repete-se o mesmo processo com p1, começando sempre do último número primo que permitiu obter resto zero. Desta forma consegue-se obter a factorização completa de p em números primos.

Por exemplo, a 24 corresponde a factorização em números primos 2^3 x 3. Uma vez que

$24 \div 2 = 12$ $\qquad$ $24 = 2 \times 12$

12 não é um número primo por isso voltamos a dividir o quociente, 12, por 2

$12 \div 2 = 6$ $\qquad$ $24 = 2 \times 2 \times 6$

6 não é um número primo por isso voltamos a dividir o quociente, 6, por 2

$6 \div 2 = 3$ $\qquad$ $24 = 2 \times 2 \times 2 \times 3 = 2^3 \times 3$

3 é um número primo podemos parar o processo.

Anexo D

Estudo D

Anexo D.1 – Parte Prática do Exame de Tecnologia da Informática

1. Dado o programa abaixo, responda às seguintes questões:

   a) Identifique as linhas onde são declaradas variáveis.

   b) Identifique o número total de bytes ocupado por cada variável.

   c) Indique o valor dos endereços, em decimal, do primeiro byte ocupado por cada variável.

   d) Existe algum ciclo? Em caso afirmativo indique as linhas onde se inicia e qual a condição de saída.

   e) Qual a diferença entre as instruções `mov al,String1[si]` e `mov al,[si]`, de forma genérica e no contexto deste programa?

   f) Explique qual o objectivo das instruções `mov al,String1[si]` e `mov String2[di],al`. Seria possível obter o mesmo efeito através de uma única instrução? Justifique a sua resposta.

   g) Qual o objectivo das instruções `inc si` e `add di,1` que surgem nas linhas 16 e 18?

   h) Indique uma instrução alternativa a `xor si,si`.

   i) Explique, de forma sintetizada, o que faz a totalidade do código apresentado.

   j) Que alterações teria de realizar se a manipulação fosse efectuada directamente na `String1` em vez do resultado ser armazenado na `String2`?

```
1:  .8086
2:  .model  small
3:  .stack  2048

4:  dados  segment    para     'data'
5:  String1    db     'aeiouaaaeioueeaeiouiiaeiouooaeiouuuaeiou,'$'
6:  String2    db     256 dup (?)
7:  dados  ends

8:  código segment    para     'code'
9:      main   proc
10:         assume cs:codigo, ds:dados

11:         mov    ax,dados
12:         mov    ds,ax

13:         mov    si,0
14:         xor    di,di

15:label1: mov    al,String1[si]
16:         inc    si
17:label3: mov    String2[di],al
18:         add    di,1
19:         cmp    al,'$'
20:         je     label2
21:         cmp    al,'e'
22:         jne    label1
23:         mov    al,String1[si]
24:         inc    si
25:         cmp    al,'i'
26:         jne    label3
27:         jmp    label1
28:label2: mov    ah,4Ch
29:         int    21h
30:     main   endp
31: código ends
32: end    main
```

2. Complete o seguinte programa em Assembly de forma a permitir efectuar a seguinte manipulação de bits no byte menos significativo de um conjunto de 10 valores de 16 bits, armazenados a partir do endereço `Conjunto1`.

$$a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0 \rightarrow a_5\ 0\ a_4\ 1\ a_4\ 0\ a_4\ a_5$$

Os valores alterados devem ser guardados a partir do endereço `Conjunto2`.

```
.8086
.model small
.stack 2048

DADOS  segment    para   'data'
   Conjunto1  dw  10  dup(2236h)
   Conjunto2  dw  10  dup(?)
DADOS  Ends

CODIGO segment    para   'code'
   Main   Proc
       assume ds:dados, cs:codigo

       mov    ax,dados
       mov    ds,ax

       xor    si,si
       mov    cx,10

ciclo: mov    ax,Conjunto1[si]
```

COMPLETAR

```
       mov    Conjunto2[si],ax
       inc    si
       inc    si
       loop   ciclo

sai:   mov    ah,4ch
       int    21h
   Main   Endp
CODIGO Ends
End    Main
```

3. Faça um programa, em Assembly, que a partir de dois vectores de 25 bytes (caracteres) cada, declarados no segmento de dados, apresente no ecrã, recorrendo a memória de vídeo, modo texto 80*25, o conteúdo desses vectores na diagonal. Os elementos dos vectores serão impressos a partir de cada extremidade da 1ª linha. O primeiro vector será impresso a partir do canto superior esquerdo e o segundo a partir do canto superior direito. NOTA: A memória de vídeo, no caso de sistemas policromáticos, tem início na localização B800h:0000h.

Anexo E

Estudo E

Anexo E.1 – Inventário de Atitudes e Comportamentos Habituais de Estudo

Este inventário apresenta situações e comportamentos descritivos das abordagens dos alunos à aprendizagem. É importante que responda com sinceridade, equacionando a sua forma habitual de estudo e não a forma como julga que deveria estudar. Interessa-nos conhecer como organiza o estudo e realiza as aprendizagens, face à generalidade das disciplinas que compõem o seu curso. As suas respostas serão tratadas com confidencialidade. Mesmo tratando-se de um questionário bastante extenso, gostaríamos que respondesse de forma conscienciosa a todos os seus itens.

INSTRUÇÃO: Assinale com uma cruz, em cada opção, o número que corresponde ao seu grau de acordo ou de desacordo com a afirmação. A sua resposta pode ir de ①(totalmente em desacordo) até ⑥(totalmente em acordo).

| Agradecemos a colaboração

I- Reportando-se às suas atitudes e formas habituais de estudo, assim como de trabalho escolar, assinale o seu grau de concordância ou de discordância em relação a cada uma das situações apresentadas.

1.	Tenho um horário pessoal de estudo devidamente organizado	min Max ①②③④⑤⑥
2.	Quando sobre um assunto há várias perspectivas, procuro estabelecer as diferenças e as semelhanças entre elas	min Max ①②③④⑤⑥
3.	Prefiro os professores que vão directos aos assuntos e não se metem por grandes desenvolvimentos	min Max ①②③④⑤⑥
4.	Só consigo entender determinadas matérias se tiver alguém a explicar-me individualmente	min Max ①②③④⑤⑥
5.	Estudo mais porque quero realizar-me profissionalmente	min Max ①②③④⑤⑥
6.	Estudo previamente os assuntos que vão ser discutidos nas aulas	min Max ①②③④⑤⑥

7.	Memorizo definições e aspectos das matérias com algum pormenor	min Max ①②③④⑤⑥
8.	Tento pensar nas ligações entre os diferentes assuntos das matérias que estudo	min Max ①②③④⑤⑥
9.	Há aspectos nas matérias do meu curso que gostaria de estudar com maior profundidade	min Max ①②③④⑤⑥
10	Esqueço a maioria das coisas que estudo depois dos testes	min Max ①②③④⑤⑥
11	A seguir às aulas costumo ler a bibliografia recomendada ou consultar os textos de apoio	min Max ①②③④⑤⑥
12	Insisto em tentar compreender as coisas que inicialmente me parecem difíceis	min Max ①②③④⑤⑥
13	Procuro entender o sentido das matérias que estudo	min Max ①②③④⑤⑥
14	Mantenho actualizado um dossier de apontamentos sobre a maioria dos assuntos que me interessam no curso	min Max ①②③④⑤⑥
15	Apenas consigo entender as matérias quando vejo exemplos/exercícios práticos	min Max ①②③④⑤⑥
16	Ao ler um artigo ou o capítulo de um livro procuro distinguir as ideias gerais das específicas	min Max ①②③④⑤⑥
17	Vou com regularidade à biblioteca para ler ou pesquisar livros e documentos	min Max ①②③④⑤⑥
18	Tenho objectivos a atingir a curto prazo com o trabalho escolar que realizo	min Max ①②③④⑤⑥
19	Extraio as minhas próprias conclusões relativamente às matérias dadas	min Max ①②③④⑤⑥
20	Mesmo quando estudo bastante, raramente obtenho bons resultados	min Max ①②③④⑤⑥
21	Procuro sequenciar as várias matérias de forma a rentabilizar os tempos de estudo para cada disciplina	min Max ①②③④⑤⑥
22	Gosto de assistir a conferências ou a debates sobre assuntos que se relacionam com o meu curso	min Max ①②③④⑤⑥
23	Ponho em causa ou questiono-me acerca de assuntos que ouvi nas aulas ou li nos livros	min Max ①②③④⑤⑥
24	No final do dia, ainda me sinto disposto a continuar a estudar se há algo que não compreendi	min Max ①②③④⑤⑥
25	Ter sucesso em algumas disciplinas do meu curso parece estar fora do meu alcance imediato	min Max ①②③④⑤⑥
26	Após ler um livro de estudo, reformulo os aspectos principais desse livro utilizando palavras minhas	min Max ①②③④⑤⑥
27	Repito aspectos das matérias até os conseguir memorizar	min Max ①②③④⑤⑥
28	Sinto falta de alguém que me ajude a orientar e organizar o meu estudo	min Max ①②③④⑤⑥
29	Esforço-me porque acho estimulante a minha área de estudos	min Max ①②③④⑤⑥
30	Faço um resumo dos factos mais importantes e memorizo-os	min Max ①②③④⑤⑥
31	Procuro sempre relacionar aquilo que estudo com o que já conheço	min Max ①②③④⑤⑥

32	Estudo diariamente para poder acompanhar as matérias que vão sendo dadas nas aulas	min Max ①②③④⑤⑥
33	Procuro fixar as definições ou as fórmulas quando as não consigo entender	min Max ①②③④⑤⑥
34	Frequento o meu curso principalmente pelo interesse pessoal nos diversos assuntos tratados	min Max ①②③④⑤⑥
35	Quando tenho dificuldades em compreender uma parte da matéria, tento perceber porquê	min Max ①②③④⑤⑥
36	Acho que algumas disciplinas do curso exigem-me mais capacidades cognitivas do que aquelas que tenho	min Max ①②③④⑤⑥
37	Passo algum do meu tempo livre a ler sobre assuntos interessantes discutidos nas aulas	min Max ①②③④⑤⑥
38	Preciso de estudar bastante mais que os meus colegas para ter sucesso no meu curso	min Max ①②③④⑤⑥
39	Transfiro soluções ou explicações de problemas anteriores para situações ou problemas novos	min Max ①②③④⑤⑥
40	Antes dos exames, elaboro uma lista dos aspectos mais importantes e tento memorizá-los	min Max ①②③④⑤⑥
41	Preparo-me para as aulas porque quero compreender melhor as matérias	min Max ①②③④⑤⑥
42	Interessa-me toda a informação ligada ao meu curso, mesmo que não se veja uma utilidade directa/imediata	min Max ①②③④⑤⑥
43	Tenho dificuldades em entender uma grande parte das matérias que estudo	min Max ①②③④⑤⑥
44	Tenho que repetir a matéria até a fixar suficientemente	min Max ①②③④⑤⑥

II- Relativamente às suas expectativas/satisfação, assinale o seu grau de concordância em relação a cada uma das afirmações abaixo indicadas.

1.	As disciplinas do curso que frequento correspondem ao que esperava	min Max ①②③④⑤⑥
2.	As matérias dadas correspondem ao que esperava	min Max ①②③④⑤⑥
3.	Os docentes correspondem ao que esperava	min Max ①②③④⑤⑥
4.	Os colegas correspondem ao que esperava	min Max ①②③④⑤⑥
5.	O ambiente geral de trabalho corresponde ao que esperava	min Max ①②③④⑤⑥
6.	A Universidade (campus, espaços, serviços, informação, …) corresponde ao que esperava	min Max ①②③④⑤⑥
7.	A minha participação no estudo/trabalho corresponde ao que esperava	min Max ①②③④⑤⑥
8.	Os equipamentos (biblioteca, material, meios informáticos,...) correspondem ao que esperava	min Max ①②③④⑤⑥

III- Assinale qual a *razão* mais importante das possíveis dificuldades de aprendizagem que sente (marque apenas uma):

☐ Falta de bases de conhecimento	☐ Falta de atenção/concentração nas aulas	☐ Método de estudo
☐ Falta de motivação	☐ Competência dos docentes	☐ Falta de sorte
☐ Dificuldades intelectuais	☐ Dificuldade dos exames	☐ Outra razão. Indique qual?
☐ Falta de esforço/persistência pessoal	☐ Dificuldades de adaptação à Universidade	_____

Curso: ☐☐☐☐☐☐☐☐☐☐☐☐☐ **Disciplina:** ☐☐☐☐☐☐☐☐☐☐☐☐

Nº Aluno: ☐☐☐☐☐☐☐☐☐☐ **Idade:** ☐☐ **Género:** ☐**F** ☐**M**

Nº Matrículas na Disciplina: ☐☐

Status: ☐**Repetente** ☐**Caloiro**

Regime de escolaridade: ☐**Estudante** ☐**Trabalhador**

OBRIGADO

Anexo F

Estudo F

Anexo F.1 – Actividade 1

1. Dado o seguinte programa:

```
#include <stdio.h>
1:  void main()
2:  {
3:      float altura,peso;
4:      printf("Indique a altura do candidato :");
5:      scanf("%f", &altura);
6:      printf("\nIndique o peso do candidato:");
7:      scanf("%f", &peso);
8:      if (altura>=1.57 && altura<=1.90)
9:          if (peso>=70 && peso<=80)
10:             printf("111");
11:         else
12:             printf("222");
13:     else
14:         if (peso>=70 && peso<=80)
15:             printf("333");
16:         else
17:             printf("444");
18: }
```

   Qual a mensagem apresentada para cada um dos seguintes pares de valores:

   a) altura=1.5 e peso=65. _____

   b) altura=1.5 e peso=70. _____

   c) altura=1.80 e peso=65. _____

   d) altura=2.10 e peso=65. _____

   e) altura=2.10 e peso=85. _____

2. Relativamente à questão anterior, quais das seguintes afirmações são verdadeiras, no contexto de todo o programa e não apenas na estrutura de selecção em que se insere:

   a) O else da linha 13 é executado quando se verifica que `altura<1.57 && altura>1.90`.

b) O else da linha 13 é executado quando se verifica que `altura<1.57 || altura>1.90`.

c) O else da linha 16 é executado quando se verifica que `peso<70 || peso>80`.

d) O else da linha 16 é executado quando se verifica que `(peso<70 || peso>80) && (altura<1.57 || altura>1.90)`.

e) O else da linha 16 é executado quando se verifica que `(peso<70 && peso>80) || (altura<1.57 && altura>1.90)`.

3. A linha 8 e 9 do programa da pergunta anterior poderiam ser fundidas numa única da seguinte forma:

```
if (altura>=1.57 && altura<=1.90 && peso>=70 && peso<=80)
```

Porém, para garantir que o mesmo comportamento seja mantido, as restantes linhas terão de sofrer algumas alterações. Qual das seguintes codificações satisfaz estes requisitos?

a)

```
#include <stdio.h>
void main()
{
    float altura,peso;
    printf("Indique a altura do candidato :");
    scanf("%f", &altura);
    printf("\nIndique o peso do candidato :");
    scanf("%f", &peso);
    if (altura>=1.57 && altura<=1.90 && peso>=70 && peso<=80)
        printf("111");
    else
        if (peso<70 || peso>80)
            printf("222");
        else
            if (altura<1.57 || altura>1.90)
                printf("333");
            else
                printf("444");
}
```

b)

```
#include <stdio.h>
void main()
{
    float altura,peso;
    printf("Indique a altura do candidato :");
    scanf("%f", &altura);
    printf("\nIndique o peso do candidato :");
    scanf("%f", &peso);
    if (altura>=1.57 && altura<=1.90 && peso>=70 && peso<=80)
        printf("111");
    else
        if (peso<70 || peso>80)
            printf("222");
```

```
        else
            if (altura>=1.57 && altura<=1.90)
                printf("333");
            else
                printf("444");
}
```

c)

```
#include <stdio.h>
void main()
{
    float altura,peso;
    printf("Indique a altura do candidato :");
    scanf("%f", &altura);
    printf("\nIndique o peso do candidato :");
    scanf("%f", &peso);
    if (altura>=1.57 && altura<=1.90 && peso>=70 && peso<=80)
        printf("111");
    else
        if (peso>=70 && peso<=80)
            printf("333");
        else
            if (altura>=1.57 && altura<=1.90)
                printf("222");
            else
                printf("444");
}
```

d)

```
#include <stdio.h>
void main()
{
    float altura,peso;
    printf("Indique a altura do candidato :");
    scanf("%f", &altura);
    printf("\nIndique o peso do candidato :");
    scanf("%f", &peso);
    if (altura>=1.57 && altura<=1.90 && peso>=70 && peso<=80)
        printf("111");
    else
        if (peso>=70 && peso<=80)
            printf("222");
        else
            if (altura<1.57 || altura>1.90)
                printf("333");
            else
                printf("444");
}
```

Anexo F.2 – Actividade 2

1. Dado o seguinte código:

```
#include <stdio.h>
#include <math.h>

int main()
{
    int num, s=0;
    printf("Qual o número? ");
    scanf("%d", &num);
    num=abs(num);
    while (num > 0){
        s = s + num % 10;
        num = num / 10;
    }
    printf("Resultado = %d", s);
    return 0;
}
```

   Indique o resultado da sua execução, supondo que o utilizador digitou 12345 para o valor de `num`:

   a) 14

   b) 15

   c) 1234

   d) 1270

   e) Nenhuma das anteriores.

2. O que acontece se, no programa anterior, se mudar a condição do `while` para `(num>=0)`.

   a) Executa indefinidamente.

   b) Dá erro de execução devido à divisão por 0.

   c) Executa mais uma vez do que o desejado, mas o resultado é igual.

   d) Nenhuma das anteriores.

3. Quais dos seguintes programas têm o mesmo resultado que o anterior para qualquer dado de entrada.

   a)

```
#include <stdio.h>
#include <math.h>

int main()
{
    int num, s=0;
```

```
    printf("Qual o número? ");
    scanf("%d",&num);
    num=abs(num);
    do{
        s = s + num % 10;
        num = num / 10;
    } while(num);
    printf("Resultado = %d", s);
    return 0;
}
```

b)

```
#include <stdio.h>
#include <math.h>

int main()
{
    int num, s=0;
    printf("Qual o número? ");
    scanf("%d",&num);
    num=abs(num);
    do{
        s = s + num % 10;
        num = num / 10;
    } while(num/10);
    printf("Resultado = %d", s);
    return 0;
}
```

c)

```
#include <stdio.h>
#include <math.h>

int main()
{
    int num=0, s=0;
    printf("Qual o número? ");
    scanf("%d",&num);
    num=abs(num);
    while (num/10){
        s = s + num % 10;
        num = num / 10;
    }
    printf("Resultado = %d", s);
    return 0;
}
```

d)

```
#include <stdio.h>
#include <math.h>

int main()
{
    int num, s=0;
    printf("Qual o número? ");
    scanf("%d", &num);
```

```
    num=abs(num);
    for(;num>0;num=num/10)
        s = s + num % 10;
    printf("Resultado = %d", s);
    return 0;
}
```

e)

```
#include <stdio.h>
#include <math.h>

int main()
{
    int num, s=0;
    printf("Qual o número? ");
    scanf("%d", &num);
    num=abs(num);
    for(;num>0;num=num%10)
        s = s + num / 10;
    printf("Resultado = %d", s);
    return 0;
}
```

Anexo F.3 – Actividade 3

1. Dado o seguinte programa:

```
1:#include <stdio.h>

2:int count = 0;

3:void teste1();
4:void teste2();

5:void main()
6:{
7:  int count = 0;

8:  for( ; count < 5; count++)
9:  {
10:     teste1();
11:     teste2();
12:     printf("\nmain%d", count);
13: }
14:}

15:void teste1()
16:{
17: printf("\nteste1   count = %d ", ++count);
18:}

19:void teste2()
20:{
21: static int count;
22: printf("\nteste2   count = %d ", ++count);
23:}
```

Assinale as respostas correctas:

a) A variável impressa na função teste1 refere-se à variável `count` definida na linha 2.

b) A variável impressa na função teste1 refere-se à variável `count` definida na linha 7.

c) A variável impressa na função teste2 refere-se à variável `count` definida na linha 2.

d) A variável impressa na função teste2 refere-se à variável `count` definida na linha 7.

e) A variável impressa na função teste2 refere-se à variável `count` definida na linha 21.

2. Dado o seguinte programa:

```
#include <stdio.h>
#define LADO 5

int i;

void linha();
void quadrado();

int main()
{
    quadrado();
    return 0;
}

void linha()
{
    for(i=0;i<LADO;i++)
        putchar('*');
}

void quadrado()
{
    for(i=0;i<LADO;i++)
    {
        linha();
        putchar('\n');
    }
}
```

Indique qual a sua saída:

a)
```
*****
*****
*****
*****
*****
```

b)
```
****
****
****
****
****
```

c)
```
*****
```

d) Nenhuma das anteriores.

3. Dado o seguinte programa:

```
#include <stdio.h>

int a=0;

void f1(int i);
int f2(int i);

int main()
{
    f1(1);
    a=f2(1);
    printf("%d\n",a);
    return 0;
}
```

```
void f1(int a)
{
    a++;
    printf("%d,",a);
}

int f2(int a)
{
    a=5;
    a++;
    printf("%d,",a);
    return a;
}
```

Indique qual a sua saída:

a) 2,6,6

b) 2,7,7

c) 1,6,0

d) 1,5,5

e) Nenhuma das anteriores.

4. Pretende-se que o código de seguida apresentado calcule a seguinte expressão. Assuma que qualquer uma das funções factorial, a) ou b), poderá ser chamada na função `main`. Escolha as diversas opções, de forma a que o programa calcule a expressão correctamente.

$$\frac{n!}{p!\,(n-p)!}$$

```
#include <stdio.h>

void main()
{
    int i,n=0, p=0, func, num, den;
    do{
        printf("introduza um valor para n:");
        scanf("%d",&n);
        printf("introduza um valor para p:");
        scanf("%d",&p);
    } while(n<0 || p<0);
    num= ESCOLHA1;
    den= ESCOLHA2;
    func=num/den;
    printf ("O resultado da função para p=%d e n=%d e %d: ",n,p,func);
}
```

a)

```
ESCOLHA3 factorial(int n){
    int i,fact=1;
    if (n==0 || n==1){
        fact=1;
        ESCOLHA4;
    }
    else
        for(i=2;i<=n;i++)
            fact=fact*i;
    ESCOLHA5;
}
```

b)

```
int factorial(int n)
{
    if( n==0 )
        return 1;
    else
        ESCOLHA6;
}
```

	Possibilidades de escolha
ESCOLHA1	n! fact(n) factorial(n) n*factorial()
ESCOLHA2	!p*factorial(n-p) fact(p)*fact(n-p) factorial(p)*factorial(n-p) factorial(p,m-p)
ESCOLHA3	double float int void
ESCOLHA4	break return return fact return factorial exit
ESCOLHA5	break return return fact return factorial exit
ESCOLHA6	return factorial(n)*factorial(n-1) return factorial(n-1) return n*factorial(n-1) return (n-1)*factorial(n)

Anexo F.4 – Actividade 4

1. Dado o seguinte programa:

```
#include <stdio.h>
void main()
{
    int a[10]={3,4,5,6,7,8},i,temp;
    temp=a[0];
    for(i=0;i<6;i++)
        a[i]=a[i+1];
    i--;
    a[i]=temp;
    for(i=0;i<10;i++)
        printf("a[%d]=%d,",i,a[i]);
}
```

Indique qual a sua saída ("?" Significa valor desconhecido):

a) a[0]=4,a[1]=5,a[2]=6,a[3]=7,a[4]=8,a[5]=3,a[6]=0,a[7]=0,a[8]=0,a[9]=0,

b) a[0]=5,a[1]=6,a[2]=7,a[3]=8,a[4]=9,a[5]=3,a[6]=0,a[7]=0,a[8]=0,a[9]=0,

c) a[0]=4,a[1]=5,a[2]=6,a[3]=7,a[4]=8,a[5]=3,a[6]=?,a[7]=?,a[8]=?,a[9]=?,

d) a[0]=5,a[1]=6,a[2]=7,a[3]=8,a[4]=9,a[5]=3,a[6]=?,a[7]=?,a[8]=?,a[9]=?,

e) Nenhuma das anteriores.

2. Para cada um dos programas seguintes, complete a expressão assinalada de forma a que cada um produza o seguinte resultado, respectivamente:

a)

a[0]=2 a[1]=4 a[2]=6 a[3]=8 a[4]=10 a[5]=12 a[6]=14 a[7]=16 a[8]=18 a[9]=20

b)

a[0] [0]=2	a[0][1]=4	a[0] [2]=6	a[0] [3]=8	a[0] [4]=10
a[1] [0]=12	a[1][1]= 14	a[1] [2]= 16	a[1] [3]= 18	a[1] [4]= 20

c)

a[0]=2	a[1]=4
a[2]=6	a[3]=8
a[4]=10	a[5]=12
a[6]=14	a[7]=16
a[8]=18	a[9]=20

d)

a[0]=2	a[5]=12
a[1]=4	a[6]=14
a[2]=6	a[7]=16
a[3]=8	a[8]=18
a[4]=10	a[9]=20

a)

```
#include <stdio.h>
#define TAM 10

void main()
{
    int i,a[TAM]= {2,4,6,8,10,12,14,16,18,20};

    for(i=0;i<TAM;i++)
        printf("a[%d]=%d\t",COMPLETAR);
```

```
    printf("\n");
}
```

b)

```
#include <stdio.h>
#define TAM1  2
#define TAM2  5

void main()
{
    int i,j,a[TAM1][TAM2]= {{2,4,6,8,10},{12,14,16,18,20}};

    for(i=0;i<TAM1;i++){
        for(j=0;j<TAM2;j++)
            printf("a[%d][%d]=%d\t", COMPLETAR);
        printf("\n");
    }
}
```

c)

```
#include <stdio.h>
#define TAM1  2
#define TAM2  5

void main()
{
    int i,j,a[TAM1*TAM2]= {2,4,6,8,10,12,14,16,18,20};
    float media;

    for(i=0;i<TAM2;i++){
        for(j=0;j<TAM1;j++){
            printf("a[%d]=%d",COMPLETAR);
            printf("\t");
        }
        printf("\n");
    }
}
```

d)

```
#include <stdio.h>
#define TAM1  2
#define TAM2  5

void main()
{
    int i,j,a[TAM1*TAM2]= {2,4,6,8,10,12,14,16,18,20};
    float media;

    for(i=0;i<TAM2;i++){
        for(j=0;j<TAM1;j++){
            printf("a[%d]=%d",COMPLETAR);
            printf("\t");
        }
        printf("\n");
    }
}
```

3. Suponha que o seguinte código tem por objectivo verificar se nalguma das linhas de um *array* de NxN todos os seus elementos são iguais, devolvendo o valor 1 caso encontre uma linha em que essa condição se verifique e 0 em situação contrária.

```
#define N 3
…
1:int lin(char t[N][N]) {
2:  int i,j,devolver=0,soma;

3:  for(i=0,soma=0; i<N; i++){
4:      for(j=0; j<N; j++){
5:          if (t[i][0]==t[i][j])
6:              soma++;
7:          else break;
8:      }
9:      if (soma==N){
10:         devolver=1;
11:         break;
12:     }
13:     soma=0;
14: }
15: return devolver;
16:}
```

Assinale as respostas correctas:

a) O parâmetro/argumento da função da linha 1 poderia ser substituído por `(char t[][N])`.

b) O parâmetro da função da linha 1 poderia ser substituído por `(char t[N][])`.

c) O parâmetro da função da linha 1 poderia ser substituído por `(char t[][])`.

d) O ciclo for iniciado na linha 3 serve para percorrer cada uma das linhas do *array*.

e) O ciclo for iniciado na linha 3 serve para percorrer cada uma das colunas do *array*, em cada linha.

f) A atribuição `soma=0` na linha 13 é desnecessária uma vez que a mesma atribuição é feita no início do ciclo externo.

g) A atribuição `soma=0` na linha 3 seria desnecessária caso se tivesse feito esta iniciação entre as linhas 2 e 3.

h) A atribuição `soma=0` na linha 3 seria desnecessária caso se tivesse feito esta iniciação entre as linhas 3 e 4.

i) As atribuições de `soma=0` poderiam ser substituídas por uma única entre as linhas 3 e 4.

j) O `break` na linha 7 é executado quando todos os elementos da linha em verificação são diferentes.

k) O `break` na linha 7 é executado quando todos os elementos da linha em verificação são iguais.

l) O `break` na linha 7 é executado quando pelo menos um dos elementos da linha em verificação é diferente dos outros.

m) Se quisesse testar a igualdade por coluna em vez de por linha bastaria alterar a linha 5 para `if (t[0][i]==t[i][j])`, mantendo todo o restante código.

n) Se quisesse testar a igualdade por coluna em vez de por linha bastaria alterar a linha 5 para `if (t[0][i]==t[j][i])`, mantendo todo o restante código.

4. Dado o seguinte programa:

```
#include <stdio.h>
#include <string.h>
#define TAM 5

int main()
{
    char x[6][6];
    int i;

    for(i=4;i>=0;i--)
        strcpy(x[i],"ABCDEF");
    strcpy(x[5],"BERNA");
    for(i=0;i<6;i++)
        printf("x[%d]=%s\n",i,x[i]);
}
```

indique o conteúdo de cada um dos elementos de x, nomeadamente:

x[0]= ***ESCOLHA***

x[1]= ***ESCOLHA***

x[2]= ***ESCOLHA***

x[3]= ***ESCOLHA***

x[4]= ***ESCOLHA***

x[5]= ***ESCOLHA***

(Para cada um dos elementos x[i] o aluno teria de escolher uma das seguintes opções: "ABCDEF", "BERNA", "BERNAF", vazio).

Anexo F.5 – Actividade 5 – Sessão1

1. Dado o seguinte código:

```
int nums[] = {2, 4, 6};
char str[] = "programacao";
int *q = nums;
char *p = str;
```

Indique a que se refere:

a) p[1]. (*O aluno teria de escolher entre p, r, s ou t.)*

b) *(p+1). (*O aluno teria de escolher entre p, r, s ou t.)*

c) *q+1. (*O aluno teria de escolher entre 1, 2, 3 ou 4.)*

d) *(q+2). (*O aluno teria de escolher entre 2, 4, 6 ou 8.)*

2. Dado o seguinte código, o que será impresso?

```
#include <stdio.h>

void main(void)
{
    int a=4, b = 4;
    float x = 1.0;
    float *pf;
    int *p;

    pf = &x;
    *pf = *pf + 2.5;
    printf("%.2f\n", x);
    printf("%.2f\n", *pf);
    p = &a;
    printf("%d\n", *p);
    *p = b * 10;
    printf("%d\n", *p);
    printf("%d\n", a);
    p = &b;
    printf("%d\n", *p);
    printf("%d\n", b);
    printf("%d\n",*p+a);
}
```

a)
3.5
3.5
4
40
40
4
4
44

b)
1.0
3.5
4
40
4
4
4
44

c) 3.5
3.5
4
40
40
4
4
8

d) 1.0
3.5
4
40
40
4
4
4
8

3. O seguinte programa tem como objectivo calcular a subtracção entre dois inteiros `a` e `b` e colocar a diferença no local referenciado por `p`. Deve também calcular a adição desses dois inteiros `a` e `b` e colocar a soma no local referenciado por `q`. Qual dos seguintes códigos poderá realizar esta funcionalidade correctamente?

a)

```
#include <stdio.h>
void calculo(int a, int b, int *p, int *q);
int main()
{
    int a, b, soma, sub;
    printf("Introduza o valor de a=");
    scanf("%d",&a);
    printf("\nIntroduza o valor de b=");
    scanf("%d",&b);
    calculo(a,b,&sub,&soma);
    printf("A soma de 'a' e 'b' = %d\n",soma);
    printf("A subtraccao de 'a' e 'b' = %d\n",sub);
}
void calculo(int a, int b, int *p, int *q)
{
    *p=a-b;
    *q=a+b;
}
```

b)

```
#include <stdio.h>
void calculo(int a, int b, int *p, int *q);
int main()
{
    int a, b, soma, sub;
    printf("Introduza o valor de a=");
    scanf("%d",&a);
    printf("\nIntroduza o valor de b=");
    scanf("%d",&b);
    calculo(a,b,sub,soma);
    printf("A soma de 'a' e 'b' = %d\n",soma);
    printf("A subtraccao de 'a' e 'b' = %d\n",sub);
}
void calculo(int a, int b, int *p, int *q)
{
    p=a-b;
    q=a+b;
}
```

c)

```
#include <stdio.h>
void calculo(int a, int b, int *p, int *q);
int main()
{
    int a, b, soma, sub;
    printf("Introduza o valor de a=");
    scanf("%d",&a);
    printf("\nIntroduza o valor de b=");
    scanf("%d",&b);
    calculo(a,b,sub,soma);
    printf("A soma de 'a' e 'b' = %d\n",soma);
    printf("A subtraccao de 'a' e 'b' = %d\n",sub);
}
void calculo(int a, int b, int *p, int *q)
{
    *p=a-b;
    *q=a+b;
}
```

d)

```
#include <stdio.h>
void calculo(int a, int b, int *p, int *q);
int main()
{
    int a, b, soma, sub;
    printf("Introduza o valor de a=");
    scanf("%d",&a);
    printf("\nIntroduza o valor de b=");
    scanf("%d",&b);
    calculo(a,b,&sub,&soma);
    printf("A soma de 'a' e 'b' = %d\n",soma);
    printf("A subtraccao de 'a' e 'b' = %d\n",sub);
}
void calculo(int a, int b, int *p, int *q)
{
    p=&a-&b;
    q=&a+&b;
}
```

4. Dado o seguinte código, qual das seguintes funções permite mostrar correctamente o conteúdo do vector *tabela*?

```
#include <stdio.h>

#define N 7
void mostraTabela(int tabela[],int tam);
int main()
{
    int tabela[]={5,5,6,8,8,9,12};
    mostraTabela(tabela, N);
}
```

a)

```
void mostraTabela(int tabela[],int tam)
{
    int i,*p;
    printf("{ ");
    for(p=tabela; p < tabela+tam-1; p++)
        printf(" %d, ",*p);
    printf("%d }\n",*p);
}
```

b)

```
void mostraTabela(int tabela[],int tam)
{
    int i,*p;
    printf("{ ");
    for(p=tabela; p < tabela+tam-1; p++)
        printf(" %d, ",&p);
    printf("%d }\n",&p);
}
```

c)

```
void mostraTabela(int *tabela,int tam)
{
    int i,*p;
    printf("{ ");
    for(p=tabela; p < tabela+tam-1; p++)
        printf(" %d, ",p);
    printf("%d }\n",p);
}
```

d)

```
void mostraTabela(int *tabela,int tam)
{
    int i,*p;
    printf("{ ");
    for(&p=&tabela; &p < &tabela+tam-1; p++)
        printf(" %d, ",p);
    printf("%d }\n",p);
}
```

Anexo F.6 – Actividade 5 – Sessão2

1. Dado o seguinte código, indique a sua saída:

```
#include <stdio.h>

char* f1(char *c)
{
    while(1)
    {
        if(*c=='\0' || *c==' ')
            break;
        putchar(*c++);
        putchar('\n');
    }
    return c;
}

void main()
{
    char st[20]="AAA    BBB", *p=st;

    while(*p != '\0')
    {
        while(*p==' ')
            p++;
        printf("%s\n",p);
        p = f1(p);
    }
}
```

a)
```
AAA    BBB
A
A
A
BBB
B
B
B
```

b)
```
AAA BBB
A
A
A
BBB
B
B
B
```

c)
```
AAABBB
A
A
A
B
B
B
```

d)
```
AAA
A
A
A
B
B
B
```

2. Dado o seguinte código, assinale as respostas correctas:

```
#include <stdio.h>
#include <stdlib.h>
float funcao(float *tab, int tam, float *maior, float *menor)
{
    float *p;
    float media;
    media = *maior = *menor = *tab;
    for(p=tab+1; p<tab+tam; p++){
        media+=*p;
```

```
        if (*p > *maior)
            *maior=*p;
        if (*p <*menor)
            *menor=*p;
    }
    media /= tam;
    return media;
}

int main()
{
    float tabela[]={91,8.45,2,45,89},media,menor,maior;
    int n=5;
    media= funcao(tabela,n,&maior,&menor);
    printf("A media dos elementos da tabela = %.2f\n",media);
    printf("O maior elemento da tabela = %.2f\n",maior);
    printf("O menor elemento da tabela = %.2f\n",menor);
}
```

a) O cabeçalho da função poderia ser substituído por `float funcao(float tab[],float *maior, int *menor)`, pois o tamanho da tabela é conhecido ao passar tab[] como argumento.

b) O cabeçalho da função poderia ser substituído por `float funcao(float tab[],int tam, float *maior, float *menor)`, pois ao passar `tab[]` como argumento apenas se está a passar um ponteiro para o início de um bloco de memória desconhecendo-se a sua dimensão.

c) O cabeçalho da função poderia ser substituído por `float *funcao(float tab[],int tam, float maior, float menor)`, desde que no final se fizesse `return &media`.

d) O cabeçalho da função poderia ser substituído por `void funcao(float *tab, int tam, float *maior, float *menor, float *media)`, retirando a instrução final respeitante ao retorno da média, alterando a chamada à função para `funcao(tabela,n,&maior,&menor,&media)` e, dentro da função usar "`*media`" em vez de "`media`".

e) Como todos os valores que se pretendem devolver são do tipo `float`, o cabeçalho da função poderia ser substituído por `float *funcao(float tab[], int tam)`, desde que se fizesse o retorno dos três valores, nomeadamente `return *media; return *maior; return *menor;`

f) Como todos os valores que se pretendem devolver são do tipo `float`, o cabeçalho da função poderia ser substituído por `float *funcao(float *tab, int tam)`, desde que se fizesse o retorno dos três valores da seguinte forma: `return *media, *maior,*menor`.

3. Pretende-se que a seguinte função receba como parâmetro um vector `v` de n números inteiros e devolva um novo vector `w`, alocado dinamicamente, cujos elementos são definidos pelas fórmulas:

   $w_o=v_o$

   $w_i=v_i+w_{i-1}$, $0<i<n$

   A função não deverá alterar o conteúdo do vector original `v`.

   Qual das seguintes funções poderá cumprir esse objectivo eficazmente?

a)

```
int* somatorios(int n, int* v)
{
    int i;
    int *w;

    w=(int*)malloc(n*sizeof(int));
    w[0]=v[0];
    for(i=1;i<n;i++)
        w[i]=v[i]+w[i-1];
    return w;
}
```

b)

```
int* somatorios(int n, int* v)
{
    int i;
    int *w;

    w=(int*)malloc(n*sizeof(int));
    *w[0]=*v[0];
    for(i=1;i<n;i++)
        *w[i]=*v[i]+*w[i-1];
    return &w;
}
```

c)

```
int* somatorios(int n, int* v)
{
    int i;
    int *w;

    w[0]=(int*)malloc(sizeof(int));
    w[0]=v[0];
    for(i=1;i< n;i++){
        w[i]=(int*)malloc(sizeof(int));
        w[i]=v[i]+w[i-1];
    }
    return w;
}
```

d)

```
int* somatorios(int n, int* v)
{
    int i;
    int *w;

    w=(int*)malloc(n*sizeof(int));
    w[0]=v[0];
    for(i=1;i<n;i++){
        w=(int*)malloc(n*sizeof(int));
        w[i]=v[i]+w[i-1];
    }
    return w;
}
```

Anexo F.7 – Actividade 6

1. Dadas as seguintes afirmações, seleccione as correctas:

   a) Pilhas e filas podem ser implementadas de forma mais eficiente através de listas ligadas do que através de vectores estáticos.

   b) Uma pilha é um conjunto de elementos no qual o primeiro elemento a entrar é o último elemento a sair.

   c) Uma pilha é um conjunto de elementos no qual o primeiro elemento a entrar é o primeiro elemento a sair.

   d) Uma fila é um conjunto de elementos no qual o primeiro elemento a entrar é o último elemento a sair.

   e) Uma fila é um conjunto de elementos no qual o primeiro elemento a entrar é o primeiro elemento a sair.

   f) Uma pilha permite a adição e remoção de elementos em qualquer ponto.

   g) Uma lista ligada permite operações sobre o elemento existente no topo.

   h) Uma pilha permite apenas operações sobre o elemento existente no topo.

   i) Numa fila as inserções e remoções de elementos são sempre realizadas na mesma extremidade.

   j) Numa fila as inserções são sempre feitas numa extremidade e as remoções na extremidade oposta.

   k) Uma lista ligada simples necessita apenas de um ponteiro para o início da lista.

   l) Uma pilha é mais facilmente implementada através de uma lista ligada simples do que através de uma lista duplamente ligada.

   m) Uma fila é mais facilmente implementada através de uma lista ligada simples do que através de uma lista duplamente ligada.

2. Considere as seguintes definições:

```
#include <stdio.h>
#include <string.h>

typedef struct expressao *pno,no;

struct expressao{
    char info;
    pno prox;
};
pno pilha;
```

A seguinte implementação, embora tenha um âmbito mais alargado (fazer a avaliação de uma expressão completa), para este exercício tem apenas como objectivo verificar se uma expressão tem o número de parênteses correctos (existem tantos '(' como ')') e se estes estão bem colocados (')' não surge antes do '(' correspondente). A expressão a testar é introduzida pelo utilizador. Alguns exemplos:

(4 + (4*(2-4)) + 4 * ((4 / 45 + 34)*5)) *expressão válida*

((3 + 5)/2)) *expressão inválida*

(3 – 4))*(5 + (6*5 - 3) *expressão inválida*

Assuma a existência das funções push e pop que colocam e retiram, respectivamente, um elemento no topo da pilha. Complete os campos em branco, de forma a cumprir o objectivo especificado:

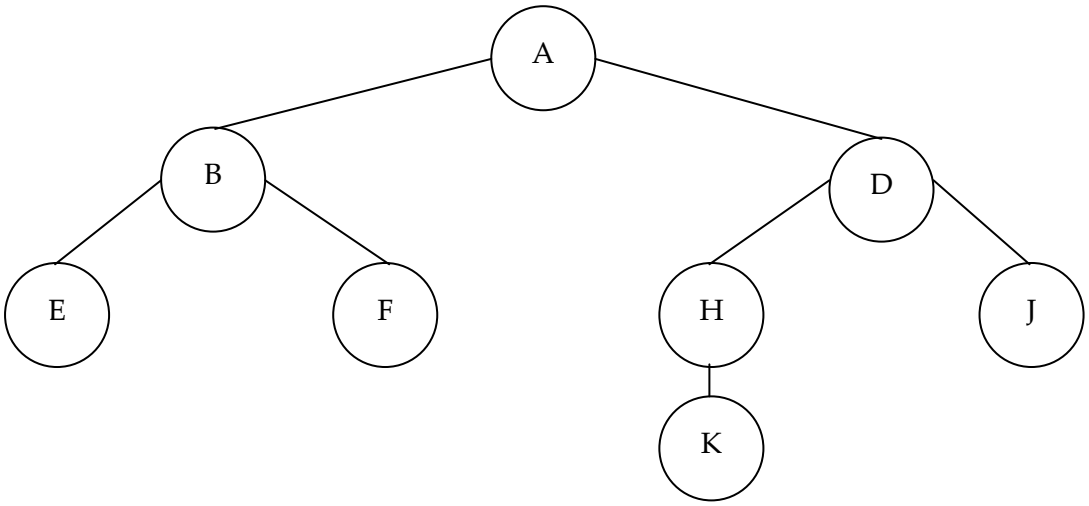
```
int pilha_vazia(void){
    if (pilha==NULL)
        return 1;
    else
        return 0;
}

pno pop(){
    pno out;
    if (pilha==NULL)
        out=NULL;
    else{
        out=pilha;
        pilha=out->prox;
    }
    return out;
}

void push(pno novo){
    novo->prox=pilha;
    pilha=novo;
}

void confere(void){
    char expressao[TAM];
    int i, valida=1;
    pno aux;

    printf("Introduza a expressao:");
    gets(expressao);

    pilha=NULL;    //Assume-se que não é necessário destruir os
                   //elementos de uma eventual pilha já existente
    for(i=0;expressao[i]!='\0' && valida;i++){
        if (expressao[i]=='('){
            aux=(ESCOLHA1)malloc(sizeof(ESCOLHA2));
            if (!aux){
                printf("Erro de alocacao de memoria\n");
                exit(1);
            }
            aux->info=expressao[i];
            ESCOLHA3;
        }
```

```
        else if (expressao[i]==')'){
            if (ESCOLHA4) {
                printf("Invalida: ) surge antes do ( correspondente\n");
                valida=0;
            } else {
                ESCOLHA5;
                ESCOLHA6;
            }
        }
    }
    if (ESCOLHA7)
        printf("Invalida: falta um ou mais ).\n");
    else
        if (valida)
            printf("Valida.\n");
}
```

	Possibilidades de escolha
ESCOLHA1	no pno *pno nenhum dos anteriores
ESCOLHA2	no pno *pno nenhum dos anteriores
ESCOLHA3	push(aux) pop(aux) pop() push(aux->info) pop(aux->info)
ESCOLHA4	pilha_vazia() !pilha_vazia() expressao[i-1]=='(' expressao[i+1]=='('
ESCOLHA5	push(aux) pop(aux) pop() push(aux->info) pop(aux->info)
ESCOLHA6	free(aux) free(aux->info) push(aux) pop(aux) pop() push(aux->info) pop(aux->info)
ESCOLHA7	valida || !pilha_vazia() !valida || pilha_vazia() valida && !pilha_vazia() !valida && pilha_vazia()

Anexo F.8 – Actividade 7

1. Analisando a árvore abaixo indique:



a) As árvores são mais flexíveis do que as listas ligadas, no que respeita à pesquisa de elementos. (*O aluno teria de escolher entre Sim ou Não.)*

b) A altura da árvore é *(O aluno teria de escolher entre 2, 3, 4, 5 ou 8)*.

c) O nó raiz é *(O aluno teria de escolher entre A, B, D, E, F, H, J ou K.)*.

d) As folhas da árvore são *(O aluno teria de escolher entre A,E,F,K,J; A,E,K,J; E,F,H,J; E,F,K,J ou "Nenhuma das anteriores".)*.

e) Os níveis dos nós B, H e K são respectivamente *(O aluno teria de escolher entre 1,2,3; 2,3,4; 3,2,1; 4,3,2.)*.

f) Os descendentes de D são *(O aluno teria de escolher entre H,J; K,J ou H,J,K.)*.

g) Os irmãos de H são *(O aluno teria de escolher entre J; E,F,J ou J,K.)*.

h) Os ascendentes de K são *(O aluno teria de escolher entre H,J; H,J,D; H,J,D,A ou H,D,A.)*.

i) A árvore é binária? *(O aluno teria de escolher entre Sim ou Não.)*

j) O resultado da travessia da árvore em "pré-ordem" é *(O aluno teria de escolher entre E,F,B,H,K,J,D,A; E,F,B,K,H,J,D,A; A,B,E,F,D,H,K,J; A,B,E,F,D,K,H,J; A,D,J,H,K,B,F,E ou A,D,J,K,H,B,F,E.)*.

k) O resultado da travessia da árvore em "pós-ordem" é *(O aluno teria de escolher entre E,F,B,H,K,J,D,A; E,F,B,K,H,J,D,A; A,B,E,F,D,H,K,J; A,B,E,F,D,K,H,J; A,D,J,H,K,B,F,E ou A,D,J,K,H,B,F,E.)*.

2. Pretende-se uma função que retorne a quantidade de folhas de uma árvore. Considere a existência da estrutura mostrada de seguida, para representar árvores de expressões.

```
struct arv{
    float valor;
    struct arv *left;
    struct arv *right;
};
typedef struct arv Arv;
```

Indique qual/quais das seguintes funções poderá/poderão cumprir essa função eficazmente:

a)

```
int folhas(Arv* a)
{
    int n = 0;
    if (a->left==NULL || a->right==NULL)
        return 1;
    n=folhas(a->left) + folhas(a->right);
    return n;
}
```

b)

```
int folhas(Arv* a)
{
    int n = 0;
    if (a->left==NULL && a->right==NULL)
        return 1;
    n=folhas(a->left) + folhas(a->right);
    return n;
}
```

c)

```
int folhas(Arv* a)
{
    int n=0;
    if (a==NULL)
        return 0;
    n=folhas(a->left) + folhas(a->right);
    if (n==0)
        return 1;
    return n;
}
```

d)

```
int folhas(Arv* a)
{
    int n=0;
    if (a==NULL)
        return 0;
    if (a->left==NULL && a->right==NULL)
        return 1;
    n=folhas(a->left) + folhas(a->right);
    return n;
}
```

3. Pretende-se uma função que verifique se duas árvores são iguais. Considere que a representação de árvores é a mesma da apresentada na pergunta anterior.

   Indique qual/quais das seguintes funções poderá/poderão cumprir essa função eficazmente:

   a)

```
int igual(Arv* a, Arv* b) {
    if (a==NULL || b==NULL)
        return 0;
    if (a==NULL && b==NULL)
        return 1;
    if (a->valor!=b->valor)
        return 0;
    return (igual(a->left,b->left) && igual(a->right,b->right));
}
```

   b)

```
int igual(Arv* a, Arv* b) {
    if (a==NULL && b==NULL)
        return 1;
    if (a==NULL || b==NULL)
        return 0;
    if (a->valor!=b->valor)
        return 0;
    return (igual(a->left,b->left) && igual(a->right,b->right));
}
```

   c)

```
int igual(Arv* a, Arv* b) {
    if (a==NULL && b==NULL)
        return 1;
    if (a==NULL || b==NULL)
        return 0;
    if (a->valor==b->valor)
        return 1;
    return (igual(a->left,b->left) && igual(a->right,b->right));
}
```

   d)

```
int igual(Arv* a, Arv* b) {
    int res=0;

    if (a!=NULL && b!=NULL) {
        res += ( a->valor==b->valor       ? 1 : 0);
        res += ( igual(a->left,b->left)   ? 1 : 0);
        res += ( igual(a->right,b->right) ? 1 : 0);
    } else if (a==NULL && b==NULL)
        res=3;

    return res / 3;
}
```

Anexo F.9 – Actividade 8

1. Suponha que se pretende criar uma estrutura Aluno que permita armazenar os campos nome e idade de um aluno, para tal, considere as seguintes alternativas:

1)
```
struct {
    char nome[50];
    int idade;
}aluno1, aluno2;
```

2)
```
struct Aluno{
    char nome[50];
    int idade;
};
struct Aluno aluno1, aluno2;
```

3)
```
struct Aluno{
    char nome[50];
    int idade;
} aluno1, aluno2;
```

4)
```
typedef struct Aluno{
    char nome[50];
    int idade;
};
Aluno aluno1, aluno2;
```

5)
```
typedef struct {
    char nome[50];
    int idade;
} Aluno;
Aluno aluno1, aluno2;
```

6)
```
typedef struct Aluno {
    char nome[50];
    int idade;
} aluno1, aluno2;
```

Considerando as definições anteriores, assinale as respostas correctas.

a) Na codificação 1), aluno1 e aluno2 são variáveis do tipo da estrutura declarada.

b) Na codificação 1), aluno1 e aluno2 são designações alternativas para o tipo da estrutura declarada.

c) A codificação 1) tem como principal desvantagem impedir a criação de novas variáveis ao longo do programa.

d) As codificações 1) e 2) são equivalentes.

e) As codificações 2) e 3) são equivalentes.

f) As codificações 1) e 3) são equivalentes.

g) As codificações 4) e 5) são equivalentes.

h) As codificações 5) e 6) são equivalentes.

i) As codificações 4) e 6) são equivalentes.

j) Na codificação 4) a estrutura está mal declarada.

k) Na codificação 5) a estrutura está mal declarada.

l) Na codificação 6) a estrutura está mal declarada.

2. Dada a seguinte declaração:

```
typedef struct aluno1 {
    char nome[50];
    int idade;
} aluno2 aluno3;
```

Indique:

a) O nome da estrutura. (*O aluno poderia escolher uma das seguintes opções: aluno1; aluno2; aluno3; Não está definido.)*

b) O nome alternativo (typedef) da estrutura. (*O aluno poderia escolher uma das seguintes opções: aluno1; aluno2; aluno3; aluno2 e aluno3.)*

c) O nome da(s) variável/variáveis. (*O aluno poderia escolher uma das seguintes opções: aluno1; aluno2; aluno3; aluno2 e aluno3.)*

3. Assumindo a estrutura Aluno referida na pergunta 1 que possui 2 campos nome e idade, indique quais das seguintes alíneas poderão constituir iniciações válidas de variáveis desse tipo:

a) struct Aluno aluno1; aluno1={“Jervásio Anacleto”,23}.

b) struct Aluno aluno1=(“Jervásio Anacleto”,23).

c) struct Aluno aluno1={“Jervásio Anacleto”,23}.

d) struct Aluno aluno1;strcpy(aluno1.nome,“Jervásio Anacleto”);aluno1.idade=23.

e) struct Aluno aluno1;aluno1.nome=”Jervásio Anacleto”;aluno1.idade=23.

f) struct Aluno aluno1;strcpy(aluno1->nome,"Jervásio Anacleto");aluno1->idade=23.

g) struct Aluno aluno1;aluno1->nome="Jervásio Anacleto";aluno1->idade=23.

h) struct Aluno aluno1[4] = { {"Joao Ratão", 20}, {"Gato das Botas", 21}, {"Branca de Neve", 18}, {"Noddy", 23} }.

i) struct Aluno[4] aluno1 = { {"Joao Ratão", 20}, {"Gato das Botas", 21}, {"Branca de Neve", 18}, {"Noddy", 23} }.

4. Suponha que o seguinte código tem por função calcular a média de idades de um *array* de estruturas Aluno. Preencha os campos assinalados com a informação em falta:

```
#include <stdio.h>
#include <string.h>

#define TAM_NOME 50
#define N 4

typedef struct{
    char nome[TAM_NOME];
    int idade;
} Aluno;

float media(Aluno *t,int tam)
{
    Aluno *p;
    float soma=0;

    for(p=t;ESCOLHA1;p++)
        soma+=ESCOLHA2;
    soma=(float)soma/tam;
    return soma;
}

void mostraAluno(Aluno *t,int tam)
{
    Aluno *p;
    for(p=t;p<t+tam;p++)
        printf("%s\t,%d\t\n",ESCOLHA3);
}

void main()
{
    Aluno tabAluno[N]={{AAA,18},{BBB,19},{CCC,19},{DDD,17}};

    mostraAluno(tabAluno,N);
    printf ("A media de idades dos alunos e %f", media(tabAluno,N));
}
```

	Possibilidades de escolha
ESCOLHA1	p<t+tam p<=t+tam p<p+tam p<=p+tam p<tam p<=tam
ESCOLHA2	p.idade p->idade *p.idade t.idade t->idade *t.idade;
ESCOLHA3	p->nome,p->idade *p.nome,*p.idade p.nome,p.idade t.nome,t.idade t->nome,t->idade *(p.nome),*(p.idade) *(t.nome),*(t.idade)

5. Suponha que se pretende uma função que permita inverter a ordem das estruturas de um ficheiro, ou seja, a 1ª passa a ser a última a 2ª a penúltima e assim sucessivamente. Indique qual das seguintes codificações poderá cumprir esse objectivo.

a)

```
void inverteFicheiro(char *fich_orig, char *fich_dest)
{
    FILE *fo, *fd;
    struct aluno al;

    if ((fo=fopen(fich_orig,"rb"))==NULL)
        printf("Erro ao abrir o ficheiro %s \n",fich_orig);
    else if ((fd=fopen(fich_dest,"wb"))==NULL) {
           printf("Erro ao abrir o ficheiro %s\n",fich_dest);
           fclose(fo);
    } else{
        fseek(fo,-1*sizeof(struct cliente),SEEK_END);
        do{
            fread(&al,sizeof(struct aluno),1,fo);
            fwrite(&al,sizeof(struct aluno),1,fd);
        } while (fseek(fo,-1*sizeof(struct aluno),SEEK_CUR)==0);
        fclose(fo);
        fclose(fd);
    }
}
```

b)

```
void inverteFicheiro(char *fich_orig, char *fich_dest)
{
    FILE *fo, *fd;
    struct aluno al;
```

```
    if ((fo=fopen(fich_orig,"rb"))==NULL)
        printf("Erro ao abrir o ficheiro %s \n",fich_orig);
    else if ((fd=fopen(fich_dest,"wb"))==NULL) {
            printf("Erro ao abrir o ficheiro %s\n",fich_dest);
            fclose(fo);
    }else{
        if (fseek(fo,-1*sizeof(struct cliente),SEEK_END)==0){
            do{
                fread(&al,sizeof(struct aluno),1,fo);
                fwrite(&al,sizeof(struct aluno),1,fd);
            }while (fseek(fo,-2*sizeof(struct aluno),SEEK_CUR)==0);
        }
        fclose(fo);
        fclose(fd);
    }
}
```

c)

```
void inverteFicheiro(char *fich_orig, char *fich_dest)
{
    FILE *fo, *fd;
    struct aluno al;

    if ((fo=fopen(fich_orig,"rb"))==NULL)
        printf("Erro ao abrir o ficheiro %s \n",fich_orig);
    else if ((fd=fopen(fich_dest,"wb"))==NULL) {
            printf("Erro ao abrir o ficheiro %s\n",fich_dest);
            fclose(fo);
    } else{
        while (fread(&al,sizeof(struct aluno),1,fo)==1) {
            fwrite(&al,sizeof(struct aluno),1,fd);
            fseek(fd,-1*sizeof(struct aluno),SEEK_CUR);
        }
        fclose(fo);
        fclose(fd);
    }
}
```

d)

```
void inverteFicheiro(char *fich_orig, char *fich_dest)
{
    FILE *fo, *fd;
    struct aluno al;
    if ((fo=fopen(fich_orig,"rb"))==NULL)
        printf("Erro ao abrir o ficheiro %s \n",fich_orig);
    else if ((fd=fopen(fich_dest,"wb"))==NULL) {
            printf("Erro ao abrir o ficheiro %s\n",fich_dest);
            fclose(fo);
    }else{
        while (fread(&al,sizeof(struct aluno),1,fo)==1) {
            fwrite(&al,sizeof(struct aluno),1,fd);
            fseek(fd,-2*sizeof(struct aluno),SEEK_CUR);
        }
        fclose(fo);
        fclose(fd);
    }
}
```

Anexo F.10 – Actividade 9

1. Seleccione as afirmações verdadeiras:

   a) Listas ligadas são estruturas de dados cujo número de elementos é fixo não podendo ser modificado com o decorrer do tempo.

   b) A utilização de *arrays* representa uma economia de espaço em memória em relação às listas ligadas.

   c) A inserção de um novo elemento no meio de uma lista ligada requer a modificação do valor de apenas dois ponteiros (do elemento anterior e do elemento seguinte).

   d) A remoção de um elemento posicionado no meio de uma lista ligada requer a modificação do valor de apenas um ponteiro.

   e) A junção de duas estruturas de dados implementadas através de um *array* é normalmente mais rápida do que a junção de duas estruturas implementadas através de listas ligadas.

   f) Numa lista ligada a procura de elementos é feita de forma sequencial, não sendo possível aceder directamente ao elemento pretendido.

   g) O aceso aos elementos de uma lista ligada é feita de forma directa.

   h) Operações de remoção de elementos em *arrays* exigem normalmente um maior processamento do que quando se utilizam listas ligadas.

   i) Numa lista ligada a inserção, remoção e troca de elementos só afectam os ponteiros (nós) vizinhos.

   j) Para remover todos os elementos de uma lista é necessário percorrer todos os elementos um a um e eliminá-los.

   k) Para remover todos os elementos de uma lista basta eliminar o elemento à cabeça da lista.

2. Assuma as seguintes declarações e função `func`, assinalando as respostas correctas:

```
typedef struct aluno *pno,no;

struct aluno{
    char nome[200];
    pno prox;
};
```

```
1: pno func(pno p, char *nome)
2: {
3:  pno actual,prev=NULL;

4:  actual=p;
5:  while(actual!=NULL && strcmp(actual->nome,nome)!=0){
6:      prev=actual;
7:      actual=actual->prox;
8:  }
9:  if (actual==NULL)
10:     return p;
11: if (prev==NULL)
12:     p=actual->prox;
13: else
14:     prev->prox=actual->prox;
15: free(actual);
16: return p;
17:}
```

a) A ordem das declarações, nas linhas 6 e 7 é arbitrária.

b) O ciclo `while` da linha 5 permite percorrer todos os elementos até encontrar o pretendido.

c) De acordo com os objectivos do programa e face à codificação apresentada, a ordem das condições do `while` é arbitrária.

d) De acordo com os objectivos do programa e face à codificação apresentada, a condição da linha 9 significa que se chegou ao final da lista e não se encontrou o elemento pretendido.

e) De acordo com os objectivos do programa e face à codificação apresentada, a condição da linha 9 significa que se chegou ao final da lista e se encontrou o elemento pretendido.

f) A condição da linha 11 testa se se está perante o 1º elemento da lista.

g) A condição da linha 11 testa se a lista tem um único elemento.

h) A execução da linha 11 significa que a lista tem pelo menos um elemento.

i) A linha 12 poderá originar um erro caso a lista tenha um único elemento.

j) A linha 14 permite libertar a memória correspondente ao elemento com a designação `actual`.

k) A linha 14 permite retirar o elemento `actual` da lista, estabelecendo uma ligação entre os seus nós vizinhos, anterior e seguinte.

l) A linha 15 permite libertar a memória correspondente ao elemento com a designação `actual`, respeitante ao elemento cujo nome se procurou.

3. Dada a seguinte estrutura de dados:

```
typedef struct dados no, *pno;
struct dados{
    int val;
    pno prox;
    pno prev;
};
```

Indique qual das seguintes funções poderá apresentar os elementos cujo valor seja igual ao valor médio dos seus vizinhos (os vizinhos de um nó são os elementos que se encontram imediatamente antes e depois dele). A função recebe como argumento um ponteiro para o início da lista.

a)

```
void mediaVizinhos(pno p){
    pno aux;
    if(p==NULL){
        printf("1\n");
        return;
    }
    else{
        if(p->prox==NULL){
            printf("2\n");
            return;
        }
        else{
            aux = p->prox;
            while(aux->prox != NULL){
                if(aux->val == (aux->prev->val + aux->prox->val)/2)
                    printf("3 - %d\n",aux->val);
                aux=aux->prox;
            }
        }
    }
}
```

b)

```
void mediaVizinhos(pno p){
    pno aux;
    if(p==NULL){
        printf("1\n");
        return;
    }
    else{
        if(p->prox==NULL){
            printf("2\n");
            return;
        }
        else{
            aux = p;
            while(aux->prox != NULL){
                if(aux->val == (aux->prev->val + aux->prox->val)/2)
                    printf("3 - %d\n",aux->val);
                aux=aux->prox;
            }
        }
    }
}
```

c)

```
void mediaVizinhos(pno p){
    pno aux;
    if(p==NULL){
        printf("1\n");
        return;
    }
    else{
        if(p->prox==NULL){
            printf("2\n");
            return;
        }
        else{
            aux = p;
            while(aux!= NULL){
                if(aux->val == (aux->prev->val + aux->prox->val)/2)
                    printf("3 - %d\n",aux->val);
                aux=aux->prox;
            }
        }
    }
}
```

d)

```
void mediaVizinhos(pno p){
    pno aux;
    if(p==NULL){
        printf("1\n");
        return;
    }
    else{
        if(p->prox==NULL){
            printf("2\n");
            return;
        }
        else{
            aux = p->prox;
            while(aux!= NULL){
                if(aux->val==(aux->prev->val + aux->prox->val)/2)
                    printf("3 - %d\n",aux->val);
                aux=aux->prox;
            }
        }
    }
}
```

Anexo G

Index of Learning Styles

Nome do aluno _____

Disponível em http://www.engr.ncsu.edu/learningstyles/ilsweb.html
Tradução para Português (Brasil) por Marcius Giorgetti.

Instruções
Faça um círculo ao redor da letra "a" ou "b" para indicar sua resposta a cada uma das questões. Por favor, assinale apenas uma alternativa para cada questão. Se as duas alternativas "a" e "b " se aplicam a você, escolha aquela que parecer mais frequente.

1. Eu compreendo melhor alguma coisa depois de
   a) experimentar.
   b) refletir sobre ela.
2. Eu me considero
   a) conservador.
   b) inovador(a).
3. Quando eu penso sobre o que fiz ontem, é mais provável que aflorem
   a) figuras.
   b) palavras.
4. Eu tendo a
   a) compreender os detalhes de um assunto, mas a estrutura geral pode ficar imprecisa.

b) compreender a estrutura geral de um assunto, mas os detalhes podem ficar imprecisos.

5. Quando estou aprendendo algum assunto novo, me ajuda
   a) falar sobre ele.
   b) refletir sobre ele.
6. Se eu fosse um professor, eu preferiria ensinar uma disciplina
   a) que tratasse com fatos e situações reais.
   b) que tratasse com idéias e teorias.
7. Eu prefiro obter novas informações através de
   a) figuras, diagramas, gráficos ou mapas.
   b) instruções escritas ou informações verbais.
8. Quando eu compreendo
   a) todas as partes, consigo entender o todo.
   b) o todo, consigo ver como as partes se encaixam.
9. Em um grupo de estudo, trabalhando um material difícil, eu provavelmente
   a) tomo a iniciativa e contribuo com idéias.
   b) assumo uma posição discreta e escuto.
10. Acho mais fácil
    a) aprender fatos.
    b) aprender conceitos.
11. Em um livro com uma porção de figuras e desenhos, eu provavelmente
    a) observo as figuras e desenhos cuidadosamente.
    b) atento para o texto escrito.
12. Quando resolvo problemas de matemática, eu
    a) usualmente trabalho de maneira a resolver uma etapa de cada vez.
    b) frequentemente antevejo as soluções, mas tenho que me esforçar muito para conceber as etapas para chegar a elas.

13. Nas disciplinas que cursei eu

    a) em geral fiz amizade com muitos dos colegas.

    b) raramente fiz amizade com muitos dos colegas.

14. Em literatura de não-ficção, eu prefiro

    a) algo que me ensine fatos novos ou me indique como fazer alguma coisa.

    b) algo que me apresente novas idéias para pensar.

15. Eu gosto de professores

    a) que colocam uma porção de diagramas no quadro.

    b) que gastam bastante tempo explicando.

16. Quando estou analisando uma estória ou novela eu

    a) penso nos incidentes e tento colocá-los juntos para identificar os temas.

    b) tenho consciência dos temas quando termino a leitura e então tenho que voltar atrás para encontrar os incidentes que os confirmem.

17. Quando inicio a resolução de um problema para casa, normalmente eu

    a) começo a trabalhar imediatamente na solução.

    b) primeiro tento compreender completamente o problema.

18. Prefiro a

    a) certeza.

    b) teoria.

19. Relembro melhor

    a) o que vejo.

    b) o que ouço.

20. É mais importante para mim que o professor

    a) apresente a matéria em etapas sequenciais claras.

    b) apresente um quadro geral e relacione a matéria com outros assuntos.

21. Eu prefiro estudar
    a) em grupo
    b) sozinho

22. Eu costumo ser considerado(a)
    a) cuidadoso(a) com os detalhes do meu trabalho.
    b) criativo(a) na maneira de realizar meu trabalho.

23. Quando busco orientação para chegar a um lugar desconhecido, eu prefiro
    a) um mapa.
    b) instruções por escrito.

24. Eu aprendo
    a) num ritmo bastante regular. Se estudar pesado, eu "chego lá".
    b) em saltos. Fico totalmente confuso(a) por algum tempo, e então, repentinamente eu tenho um "estalo ".

25. Eu prefiro primeiro
    a) experimentar as coisas.
    b) pensar sobre como é que eu vou fazer.

26. Quando estou lendo como lazer, eu prefiro escritores que
    a) explicitem claramente o que querem dizer.
    b) digam as coisas de maneira criativa, interessante.

27. Quando vejo um diagrama ou esquema em uma aula, relembro mais facilmente
    a) a figura.
    b) o que o professor disse a respeito dela.

28. Quando considero um conjunto de informações, provavelmente eu
    a) presto mais atenção nos detalhes e não percebo o quadro geral.
    b) procuro compreender o quadro geral antes de atentar para os detalhes.

29. Relembro mais facilmente

    a) algo que fiz.

    b) algo sobre o que pensei bastante.

30. Quando tenho uma tarefa para executar, eu prefiro

    a) dominar uma maneira para a execução da tarefa.

    b) encontrar novas maneiras para a execução da tarefa.

31. Quando alguém está me mostrando dados, eu prefiro

    a) diagramas ou gráficos.

    b) texto sumarizando os resultados.

32. Quando escrevo um texto, eu prefiro trabalhar (pensar a respeito ou escrever)

    a) a parte inicial do texto e avançar ordenadamente.

    b) diferentes partes do texto e ordená-las depois.

33. Quando tenho que trabalhar em um projeto em grupo, eu prefiro que se faça primeiro

    a) um debate em grupo (*brainstorming*), onde todos contribuam com idéias.

    b) a produção de propostas individuais, seguido de reunião do grupo para comparar as idéias.

34. Considero um elogio chamar alguém de

    a) sensível.

    b) imaginativo.

35. Das pessoas que conheço em uma festa, provavelmente eu me recordo melhor

    a) da sua aparência.

    b) do que eles disseram sobre si mesmas.

36. Quando estou aprendendo um assunto novo, eu prefiro

    a) concentrar-me no assunto, aprendendo o máximo possível.

    b) tentar estabelecer conexões entre o assunto e outros com ele relacionados.

37. Mais provavelmente sou considerado(a)
    a) expansivo(a).
    b) reservado(a).

38. Prefiro disciplinas que enfatizam
    a) material concreto (fatos, dados).
    b) material abstrato (conceitos, teorias).

39. Para entretenimento, eu prefiro
    a) assistir televisão.
    b) ler um livro.

40. Alguns professores iniciam suas preleções com um resumo do que irão cobrir. Tais resumos são
    a) de alguma utilidade para mim.
    b) muito úteis para mim.

41. A idéia de fazer o trabalho de casa em grupo, com a mesma nota para todos do grupo,
    a) me agrada.
    b) não me agrada.

42. Quando estou fazendo cálculos longos
    a) tendo a repetir todos os passos e conferir meu trabalho cuidadosamente.
    b) acho cansativo conferir o meu trabalho e tenho que me esforçar para fazê-lo.

43. Tendo a descrever os lugares onde estive
    a) com facilidade e com bom detalhamento.
    b) com dificuldade e sem detalhamento.

44. Quando estou resolvendo problemas em grupo, mais provavelmente eu
    a) penso nas etapas do processo de solução.
    b) penso nas possíveis conseqüências, ou sobre a aplicações da solução para uma ampla faixa de áreas.

Avaliação do questionário

O questionário

a) foi facilmente compreendido e respondi a todas as perguntas.
b) foi facilmente compreendido, mas não estava animado para respondê-lo.
c) não foi facilmente compreendido e não consegui respondê-lo.
d) não foi facilmente compreendido, mas consegui respondê-lo.

Anexo H

Teste de Dehnadi e Bornat

Test created by Saeed Dehnadi and Richard Bornat, Middlesex University

Researcher: .....

Name:	
Student No:	
Age:	
Gender:	M ☐ F ☐
A-Level or any equivalent subjects:	
Have you ever written a computer program in any language? **If so, in what language(s)?**	
Will this be your first course in programming? **If not, what other programming course(s) have you studied?** **-- and did you pass or fail?**	

The results of this test will be recorded in a database. It will never be revealed to any person who could in any way identify you from the data given above. The results will not be used for assessment purposes.

I consent to the use of the following questionnaire for the research project

1. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int a = 10;`
`int b = 20;`
 	
`a = b;`
	The new values of a and b are:
	
	☐ `a = 10 b = 10`
	☐ `a = 30 b = 20`
	☐ `a = 0 b = 10`
	☐ `a = 20 b = 20`
	☐ `a = 0 b = 30`
	☐ `a = 10 b = 20`
	☐ `a = 20 b = 10`
	☐ `a = 20 b = 0`
	☐ `a = 10 b = 30`
	☐ `a = 30 b = 0`
	
	Any other values for a and b:
	
	a = **b** =
	a = **b** =
	a = **b** =
		Use this column for your rough notes please

2. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int a = 10;`
`int b = 20;`
 	
`b = a;`
	The new values of a and b are:
	
	☐ `a = 0 b = 30`
	☐ `a = 30 b = 10`
	☐ `a = 0 b = 10`
	☐ `a = 20 b = 0`
	☐ `a = 20 b = 20`
	☐ `a = 20 b = 10`
	☐ `a = 30 b = 0`
	☐ `a = 10 b = 20`
	☐ `a = 10 b = 10`
	☐ `a = 10 b = 30`
	
	Any other values for a and b:
	
	`a = b =`
	`a = b =`
	`a = b =`	

3. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int big = 10;`
`int small = 20;`
 	
`big = small;`
	The new values of big and small are:
	
	☐ `big = 30 small = 0`
	☐ `big = 20 small = 0`
	☐ `big = 0 small = 30`
	☐ `big = 20 small = 10`
	☐ `big = 10 small = 10`
	☐ `big = 30 small = 20`
	☐ `big = 20 small = 20`
	☐ `big = 0 small = 10`
	☐ `big = 10 small = 20`
	☐ `big = 10 small = 30`
	
	Any other values for big and small :
	
	`big = small =`
	`big = small =`
	`big = small =`
		Use this column for your rough notes please

4. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int a = 10;`
`int b = 20;`
 	
`a = b;`
`b = a;`
	The new values of a and b are:
	
	☐ `a = 10 b = 0`
	☐ `a = 10 b = 10`
	☐ `a = 30 b = 50`
	☐ `a = 0 b = 20`
	☐ `a = 40 b = 30`
	☐ `a = 30 b = 0`
	☐ `a = 20 b = 20`
	☐ `a = 0 b = 30`
	☐ `a = 30 b = 30`
	☐ `a = 10 b = 20`
	☐ `a = 20 b = 10`
	
	Any other values for a and b:
	`a = b =`
	`a = b =`
	`a = b =`	

5. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int a = 10;`
`int b = 20;`
 	
`b = a;`
`a = b;`
	The new values of a and b are:
	
	☐ `a = 30 b = 50`
	☐ `a = 10 b = 10`
	☐ `a = 20 b = 20`
	☐ `a = 10 b = 0`
	☐ `a = 0 b = 20`
	☐ `a = 30 b = 0`
	☐ `a = 40 b = 30`
	☐ `a = 0 b = 30`
	☐ `a = 20 b = 10`
	☐ `a = 30 b = 30`
	☐ `a = 10 b = 20`
	
	Any other values for a and b:
	`a = b =`
	`a = b =`
	`a = b =`	

6. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int a = 10;`
`int b = 20;`
`int c = 30;`
 	
`a = b;`
`b = c;`
	The new values of a and b and c are:
	
	☐ a = 30 b = 50 c = 30
	☐ a = 60 b = 0 c = 0
	☐ a = 10 b = 30 c = 40
	☐ a = 0 b = 10 c = 0
	☐ a = 10 b = 10 c = 10
	☐ a = 60 b = 20 c = 30
	☐ a = 30 b = 50 c = 0
	☐ a = 20 b = 30 c = 0
	☐ a = 10 b = 20 c = 30
	☐ a = 20 b = 20 c = 20
	☐ a = 0 b = 10 c = 20
	☐ a = 20 b = 30 c = 30
	☐ a = 10 b = 10 c = 20
	☐ a = 30 b = 30 c = 50
	☐ a = 0 b = 30 c = 50
	☐ a = 30 b = 30 c = 30
	☐ a = 0 b = 0 c = 60
	☐ a = 20 b = 30 c = 20
	
	Any other values for a and b and c :
	
	a = b = c =
	a = b = c =
	a = b = c =
		Use this column for your rough notes please

7. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int a = 5;`
`int b = 3;`
`int c = 7;`
 	
`a = c;`
`b = a;`
`c = b;`
	The new values of a and b and c are:
	
	☐ a = 3 b = 5 c = 5
	☐ a = 3 b = 3 c = 3
	☐ a = 12 b = 14 c = 22
	☐ a = 8 b = 15 c = 12
	☐ a = 7 b = 7 c = 7
	☐ a = 5 b = 3 c = 7
	☐ a = 5 b = 5 c = 5
	☐ a = 7 b = 5 c = 3
	☐ a = 3 b = 7 c = 5
	☐ a = 12 b = 8 c = 10
	☐ a = 10 b = 8 c = 12
	☐ a = 0 b = 0 c = 7
	☐ a = 0 b = 0 c = 15
	☐ a = 3 b = 12 c = 0
	☐ a = 3 b = 5 c = 7
	
	Any other values for a and b and c :
	
	a = b = c =
	a = b = c =
	a = b = c =	

8. Read the following statements and tick the box next to the correct answer in the next column.	**The new values of a and b and c are:**	**Use this column for your rough notes please**

`int a = 5;`
`int b = 3;`
`int c = 7;`
 	
`c = b;`
`b = a;`
`a = c;`
	☐ a = 3 b = 5 c = 7
	☐ a = 15 b = 10 c = 22
	☐ a = 12 b = 8 c = 10
	☐ a = 7 b = 7 c = 7
	☐ a = 3 b = 5 c = 3
	☐ a = 0 b = 0 c = 7
	☐ a = 5 b = 3 c = 7
	☐ a = 3 b = 3 c = 3
	☐ a = 7 b = 5 c = 3
	☐ a = 3 b = 5 c = 0
	☐ a = 3 b = 7 c = 5
	☐ a = 8 b = 10 c = 12
	☐ a = 5 b = 5 c = 5
	☐ a = 15 b = 8 c = 10
	☐ a = 10 b = 5 c = 0
	☐ a = 0 b = 0 c = 15
	
	Any other values for a and b and c :
	
	a = b = c =
	a = b = c =
	a = b = c =	

9. Read the following statements and tick the box next to the correct answer in the next column.	**The new values of a and b and c are:**	

`int a = 5;`
`int b = 3;`
`int c = 7;`
 	
`c = b;`
`a = c;`
`b = a;`
	☐ a = 15 b = 18 c = 10
	☐ a = 7 b = 5 c = 3
	☐ a = 7 b = 0 c = 5
	☐ a = 0 b = 3 c = 0
	☐ a = 10 b = 0 c = 5
	☐ a = 5 b = 3 c = 7
	☐ a = 3 b = 3 c = 3
	☐ a = 12 b = 8 c = 10
	☐ a = 7 b = 7 c = 7
	☐ a = 15 b = 10 c = 12
	☐ a = 7 b = 7 c = 5
	☐ a = 8 b = 10 c = 12
	☐ a = 0 b = 15 c = 0
	☐ a = 7 b = 3 c = 5
	☐ a = 5 b = 5 c = 5
	☐ a = 3 b = 7 c = 5
	
	Any other values for a and b and c :
	
	a = b = c =
	a = b = c =
	a = b = c =	

10. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int a = 5;`
`int b = 3;`
`int c = 7;`
 	
`b = a;`
`c = b;`
`a = c;`
	The new values of a and b and c are:
	
	☐ a = 0 b = 7 c = 3
	☐ a = 12 b = 8 c = 10
	☐ a = 15 b = 0 c = 0
	☐ a = 0 b = 7 c = 8
	☐ a = 3 b = 7 c = 3
	☐ a = 5 b = 3 c = 7
	☐ a = 3 b = 3 c = 3
	☐ a = 7 b = 5 c = 3
	☐ a = 20 b = 8 c = 15
	☐ a = 3 b = 7 c = 5
	☐ a = 5 b = 0 c = 0
	☐ a = 8 b = 10 c = 15
	☐ a = 5 b = 5 c = 5
	☐ a = 8 b = 10 c = 12
	☐ a = 5 b = 7 c = 3
	☐ a = 7 b = 7 c = 7
	
	Any other values for a and b and c :
	
	a = b = c =
	a = b = c =
	a = b = c =
		Use this column for your rough notes please

11. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int a = 5;`
`int b = 3;`
`int c = 7;`
 	
`b = a;`
`a = c;`
`c = b;`
	The new values of a and b and c are:
	
	☐ a = 8 b = 18 c = 15
	☐ a = 7 b = 0 c = 8
	☐ a = 5 b = 5 c = 5
	☐ a = 12 b = 8 c = 15
	☐ a = 7 b = 0 c = 5
	☐ a = 3 b = 7 c = 5
	☐ a = 7 b = 5 c = 3
	☐ a = 0 b = 15 c = 0
	☐ a = 0 b = 3 c = 0
	☐ a = 3 b = 3 c = 3
	☐ a = 7 b = 7 c = 7
	☐ a = 12 b = 8 c = 10
	☐ a = 8 b = 10 c = 12
	☐ a = 7 b = 5 c = 5
	☐ a = 5 b = 3 c = 7
	☐ a = 7 b = 3 c = 5
	
	Any other values for a and b and c :
	
	a = b = c =
	a = b = c =
	a = b = c =	

12. Read the following statements and tick the box next to the correct answer in the next column.	**The new values of a and b and c are:**	**Use this column for your rough notes please**

`int a = 5;`
`int b = 3;`
`int c = 7;`
 	
`a = c;`
`c = b;`
`b = a;`
	☐ `a = 0 b = 12 c = 3`
	☐ `a = 5 b = 5 c = 5`
	☐ `a = 0 b = 7 c = 3`
	☐ `a = 8 b = 10 c = 12`
	☐ `a = 15 b = 0 c = 0`
	☐ `a = 3 b = 7 c = 5`
	☐ `a = 12 b = 15 c = 10`
	☐ `a = 5 b = 7 c = 3`
	☐ `a = 3 b = 3 c = 3`
	☐ `a = 7 b = 7 c = 7`
	☐ `a = 12 b = 8 c = 10`
	☐ `a = 5 b = 0 c = 0`
	☐ `a = 5 b = 3 c = 7`
	☐ `a = 7 b = 7 c = 3`
	☐ `a = 20 b = 15 c = 12`
	☐ `a = 7 b = 5 c = 3`
	
	Any other values for a and b and c :
	
	`a = b = c =`
	`a = b = c =`
	`a = b = c =`	

Anexo I

Componente LEGO

Descrição

Suponha a existência de uma linguagem LEGO que permita criar construções utilizando peças LEGO de 2x2 e que apenas se podem utilizar peças de 4 cores diferentes. Assumamos que as cores utilizadas são vermelho (V), amarelo (A), laranja (L) e preto (P). Suponha ainda que dispõe de um tabuleiro de 16x16, que servirá de base para colocação das peças. Admita que a linguagem LEGO proposta utiliza para a codificação das instruções apenas peças de 1x1, sendo cada instrução codificada à custa de 5 peças, da seguinte forma:

1 peça de 1x1 para indicar a cor da peça de 2x2 a colocar no tabuleiro (assuma que a cor da peça de codificação é igual à da peça de construção);

2 peças de 1x1 para indicar a coordenada x da peça de 2x2 a colocar no tabuleiro;

2 peças de 1x1 para indicar a coordenada y da peça de 2x2 a colocar no tabuleiro;

Assuma que todas as peças utilizadas na construção são do mesmo tipo (2x2), o que faz com que não seja necessário especificar o tipo de peça. Por outro lado, a utilização de peças quadradas evita a especificação da orientação da peça.

Uma vez que se utilizam 2 peças para representar 16 posições e que apenas dispomos de 4 cores (V, A, L e P) para a sua codificação/representação, assumamos as seguintes combinações de cores:

	V	A	L	P
V	0	1	2	3
A	4	5	6	7
L	8	9	10	11
P	12	13	14	15

Exercício 1

Com base nas regras da linguagem LEGO apresentada, descodifique o "programa" apresentado na figura seguinte. Assuma que o canto superior esquerdo do tabuleiro de construção (base azul) é representado pelas coordenadas (0,0) e o canto inferior direito é representado pelas coordenadas (15,15).

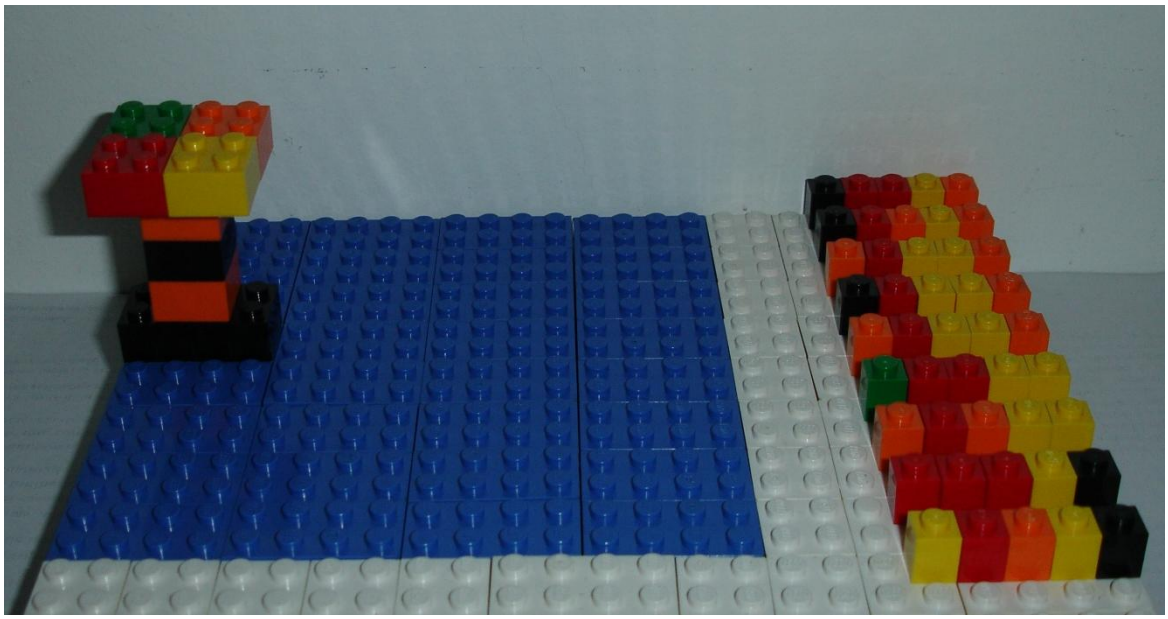


Figura I.1: Descodificação de um programa na linguagem LEGO

Sugestão

Considere que cada acção, peça 2x2 colocada no tabuleiro, é representada por 5 peças (1 para a representação da cor, 2 para a representação da posição x e 2 para a representação da posição y).

Exercício 2

Com base nas regras da linguagem LEGO referida, apresente o conjunto de instruções necessárias para criar um "programa" que ilustre a letra C, constituída por peças 2x2 com as cores amarelo e preto, alternadas. O canto superior esquerdo da letra "C" deverá ficar nas coordenadas (2,8) do tabuleiro de construção.

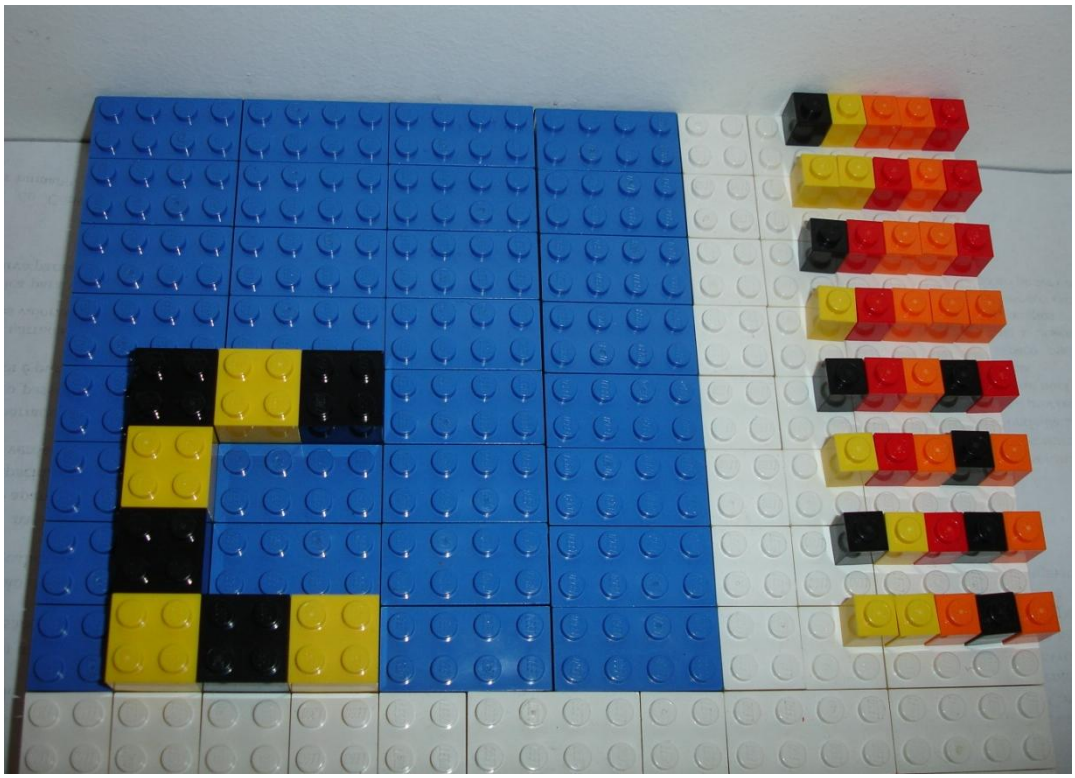


Figura I.2: Construção de um programa na linguagem LEGO

Anexo J

Scratch

O Scratch é uma ferramenta em desenvolvimento pelo grupo de investigação *Lifelong Kindergarten* do *MIT Media Lab*, em colaboração com o *KIDS research group do UCLA Graduate School of Education & Information Studies*. É um "ambiente de programação" que permite facilmente criar histórias interactivas, jogos, animações, entre outros, com diferentes graus de complexidade. Estas criações podem ainda ser partilhadas na *world wide web*. A interface geral, ilustrada na figura J.1, é constituída principalmente por quatro áreas, Paleta de Blocos, Área de *Script*, Palco e Lista de *Sprites*. Os projectos Scratch são constituídos por objectos designados *sprites***.** Um *sprite* é um gráfico que pode produzir efeitos animados ou ser alvo de acções (Comandos ou Sons). É possível escolher diferentes *sprites* da biblioteca do Scratch ou desenhar *sprites* originais. Os *sprites* podem ser alvo de variadas acções, por exemplo, pode-se mudar a aparência de um *sprite* atribuindo-lhe diferentes "Trajes". É possível dar instruções a um *sprite* para este se mover para determinado sítio ou tocar determinada música ou reagir a outros *sprites*. Para dizer a um *sprite* o que fazer, há que empilhar blocos gráficos designados scripts. Quando se executa (*double-click*) um script o Scratch executa os blocos de acordo com a sua ordem e funcionalidade. Para programar determinada acção há que escolher o *sprite* que se quer programar, seleccionar o separador Script e construir a codificação na área de *Script* (arrastar os blocos de programação de cada uma das áreas de interesse para a área de *script*).

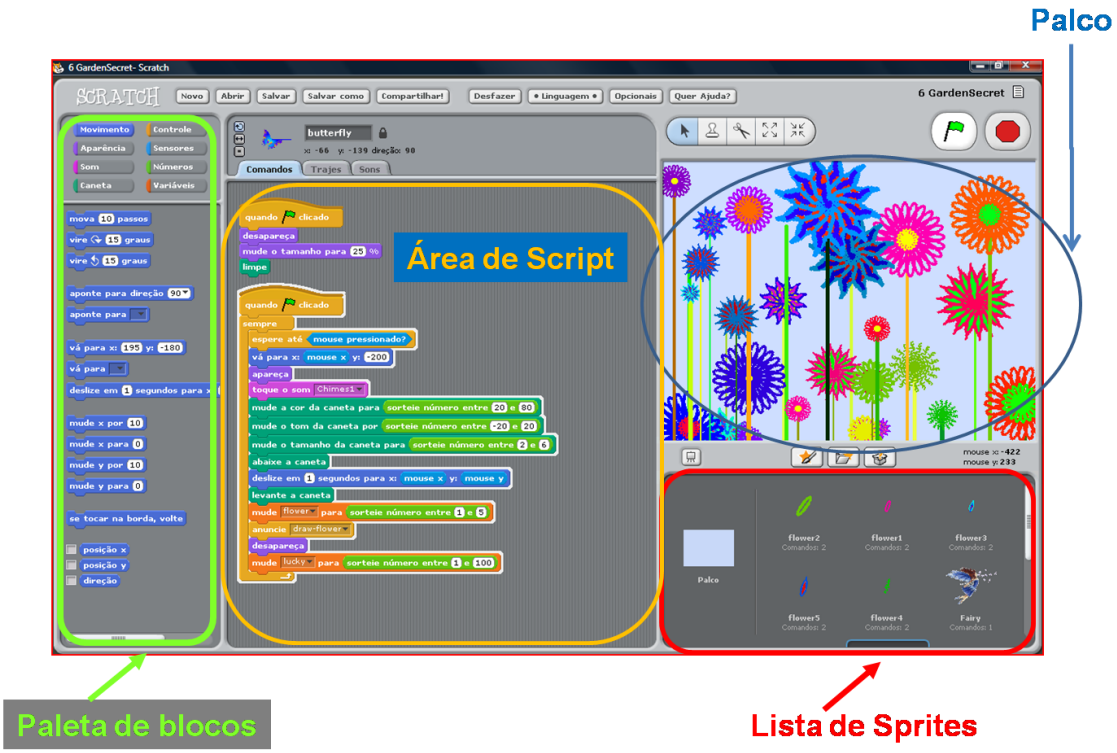


Figura J.1: Scratch – Interface geral

Os blocos de programação estão organizados em oito categorias identificadas por cores diferentes, como apresentado na figura J.2.

Figura J.2: Scratch – Blocos de programação

Os blocos de Movimento permitem mover, virar ou apontar o *sprite* para determinada posição. Os blocos de Aparência permitem alterar o *background* ou os trajes/aparência do *sprite*. Os blocos de Som permitem tocar notas ou instrumentos. Os blocos Caneta permitem realizar acções com características de desenho. Os blocos de Controlo, de grande relevância para desenvolvimento de capacidades de programação, possuem variadas estruturas que permitem controlar acções, nomeadamente com carácter de selecção e repetição. Os blocos Sensores permitem detectar comportamentos do tipo: se um *sprite* toca noutro *sprite* ou se o rato foi pressionado. Os blocos Números permitem realizar operações aritméticas e lógicas. Os blocos Variáveis permitem criar e armazenar valores. A tabela J.1, tabela J.2, tabela J.3,

tabela J.4 e tabela J.5 apresentam exemplos de blocos de programação de algumas das categorias referidas.

	Movimento
	Movimenta o sprite para a frente e para trás. Roda o sprite no sentido dos ponteiros do relógio. Roda o sprite no sentido contrário ao dos ponteiros do relógio.

Tabela J.1: Scratch – Exemplos de blocos de movimento

	Controlo
	Executa o script quando a bandeira for clicada. Executa o script quando o Sprite1 for clicado. Executa o bloco no seu interior infinitamente. Executa o bloco no seu interior um número específico de vezes. Executa o bloco no seu interior até a condição especificada ser verdadeira. Executa o bloco no seu interior se a condição especificada for verdadeira.

Tabela J.2: Scratch – Exemplos de blocos de Controlo

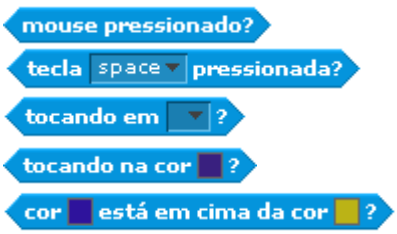
	Sensores
	Devolve verdadeiro se o botão do rato for pressionado. Devolve verdadeiro se a tecla especificada for pressionada. Devolve verdadeiro se o sprite estiver a tocar num elemento. Devolve verdadeiro se o sprite estiver a tocar na cor especificada Devolve verdadeiro se a 1ª cor estiver a tocar na 2ª cor.

Tabela J.3: Scratch – Exemplos de blocos de Sensores

	Números
	Adiciona dois números.
	Subtrai dois números.
	Multiplica dois números.
	Divide o 1º número pelo 2º.
	Devolve verdadeiro se o 1º valor for menor do que o 2º.
	Devolve verdadeiro se os dois valores forem iguais.
	Devolve verdadeiro se o 1º valor for maior do que o 2º.
	Devolve verdadeiro se ambas as condições forem verdadeiras.
	Devolve verdadeiro se alguma das condições for verdadeira.
	Nega a condição.
	Devolve o resultado da função seleccionada (abs, sqrt, sin, cos, tan, asin, acos, atan, ln, log, e^, 10^) ao valor especificado.
	Devolve o resto da divisão do 1º número pelo 2º.
	Devolve um número aleatório entre os valores especificados.

Tabela J.4: Scratch – Exemplos de blocos Números

	Variáveis
	Permite criar e nomear uma variável.
	Apaga todos os blocos associados a uma variável.
	Altera o valor da variável para o valor especificado.
	Muda o valor da variável pela quantidade especificada.

Tabela J.5: Scratch – Exemplos de blocos Variáveis

Proposta de actividades

O professor poderá explicar a filosofia geral do Scratch, e indicar aos alunos que realizem acções de forma a colocarem um "actor" à escolha em movimento, atribuindo-lhes comportamentos a gosto. Posteriormente, de forma a certificar-se que os alunos perceberam os conceitos de movimentação, o professor poderá propor a realização de acções mais específicas, por exemplo, pedindo ao aluno que mova o "actor", na horizontal ou numa diagonal, o que poderá ser feito através das acções apresentadas na figura J.3.

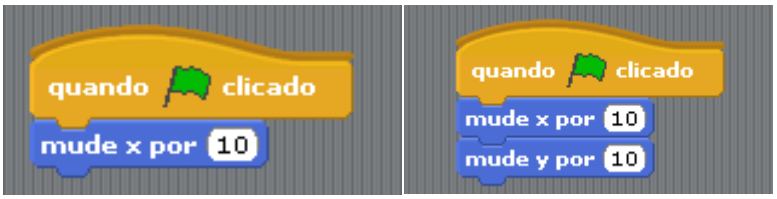


Figura J.3: Scratch – Exemplos de codificações de movimento

O Professor poderá aqui introduzir o conceito de variável, para referir um valor que não é fixo mas que vai variando em função de determinado valor, o que poderá ser feito através da codificação apresentada na figura J.4.

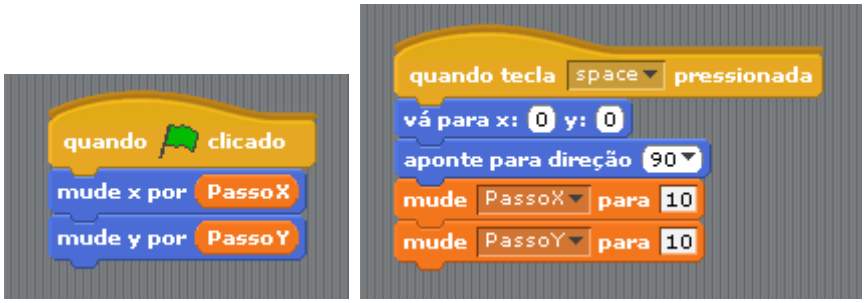


Figura J.4: Scratch – Exemplos de codificações de movimento com variáveis

Neste caso o aluno poderá visualizar e variar, no palco, os valores das variáveis e verificar os efeitos produzidos.

Com os exemplos anteriores o aluno verificará que após vários *clicks* o *sprite*/actor poderá desaparecer do palco, pelo que o professor poderá aproveitar para explicar a utilização de Sensores e convidar o aluno a descobrir uma condição para que o actor nunca desapareça, o que poderá ser feito, por exemplo, através da codificação apresentada na figura J.5.

Figura J.5: Scratch – Exemplo1 de uma codificação de movimento com variáveis e selecções

Através do exemplo muito simples, apresentado na figura J.6, poderá ser introduzido também o conceito de repetição, mostrando a execução de uma acção um número finito ou aparentemente infinito de vezes.

Figura J.6: Scratch – Exemplo2 de uma codificação de movimento com variáveis, selecções e repetições

O professor poderá sugerir aos alunos pequenas alterações à codificação anterior, de forma a produzir o resultado apresentado na figura J.7.

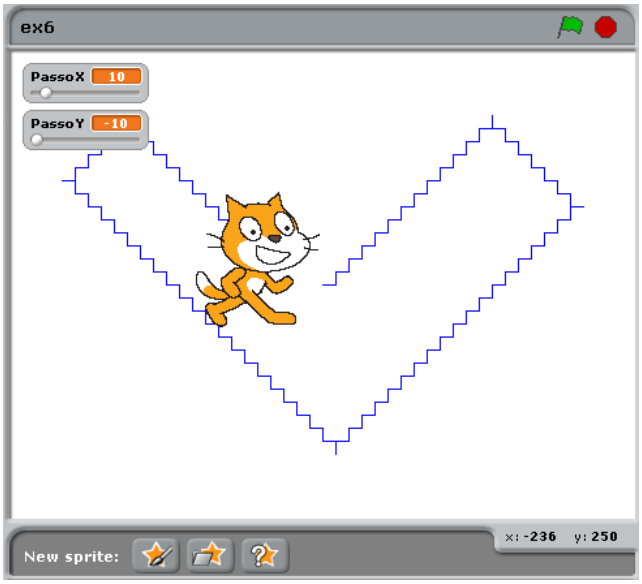


Figura J.7: Scratch – Saída de uma codificação (exemplo3) de movimento com variáveis, selecções e repetições

O que poderá ser feito recorrendo à codificação apresentada na figura J.8.

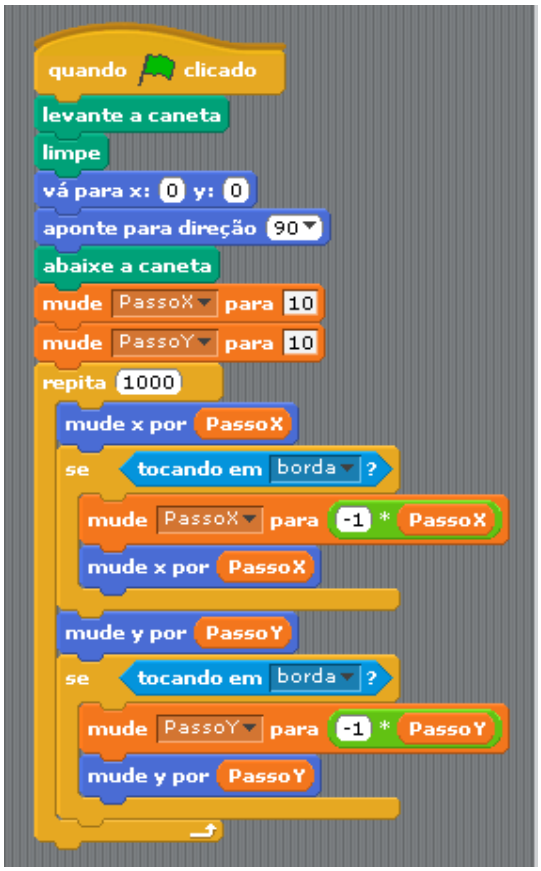


Figura J.8: Scratch – Codificação (exemplo3) de movimento com variáveis, selecções e repetições

Ou ainda, o resultado apresentado na figura J.9.

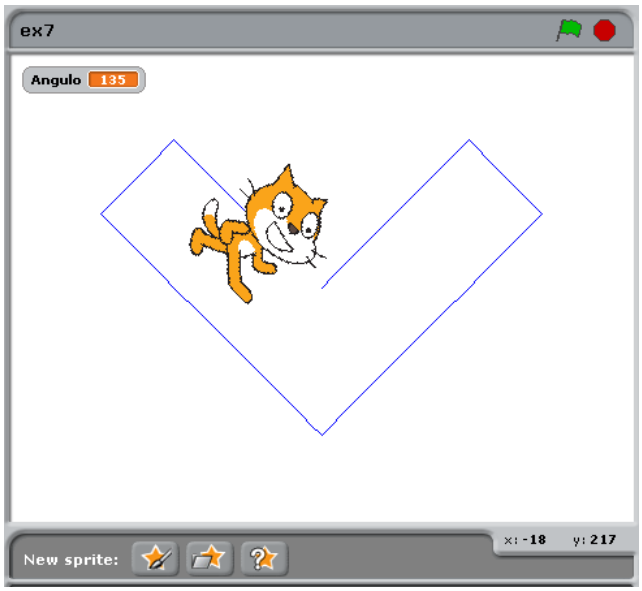


Figura J.9: Scratch – Saída de uma codificação (exemplo4) de movimento com variáveis, selecções e repetições

O que poderá ser feito recorrendo à codificação apresentada na figura J.10.

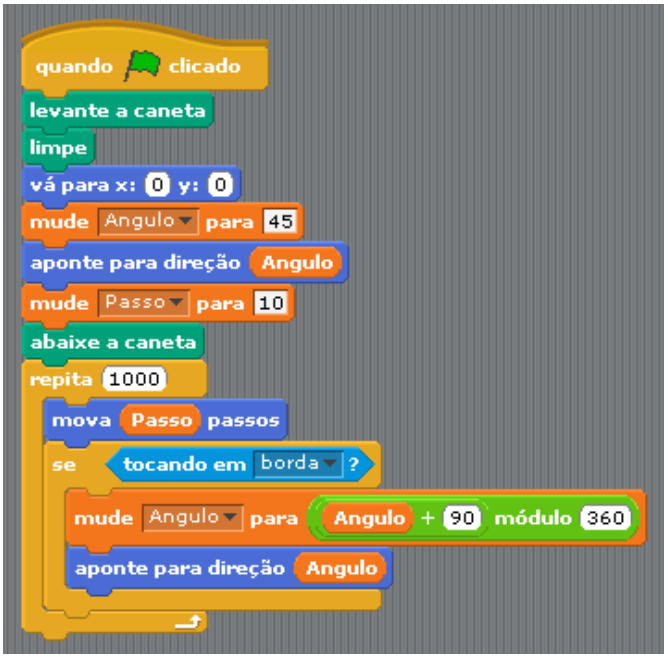


Figura J.10: Scratch – Codificação (exemplo4) de movimento com variáveis, selecções e repetições

Sugere-se que inicialmente existam actividades mais demonstrativas e acompanhadas pelo professor. Posteriormente, aconselha-se um conjunto de actividades, em que a intervenção do professor seja menor. Assim, sugere-se que o professor proponha a realização de actividades de entendimento, pedindo aos alunos para interpretarem determinado

código. Por exemplo, preverem, sem executarem, o comportamento do código apresentado na figura J.11.

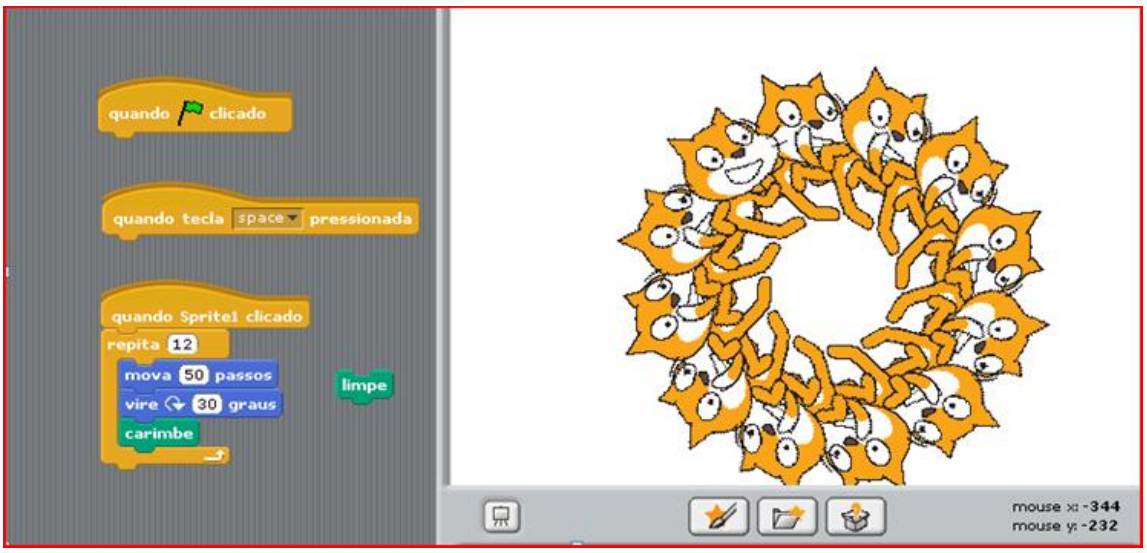


Figura J.11: Scratch – Codificação e saída (exemplo5) de movimento com variáveis e repetições

O professor poderá propor actividades de continuação das tarefas anteriores, consistindo em pedir aos alunos para alterarem comportamentos. Por exemplo, o professor poderá pedir a alteração da codificação anterior, para o comportamento expresso na figura J.12.

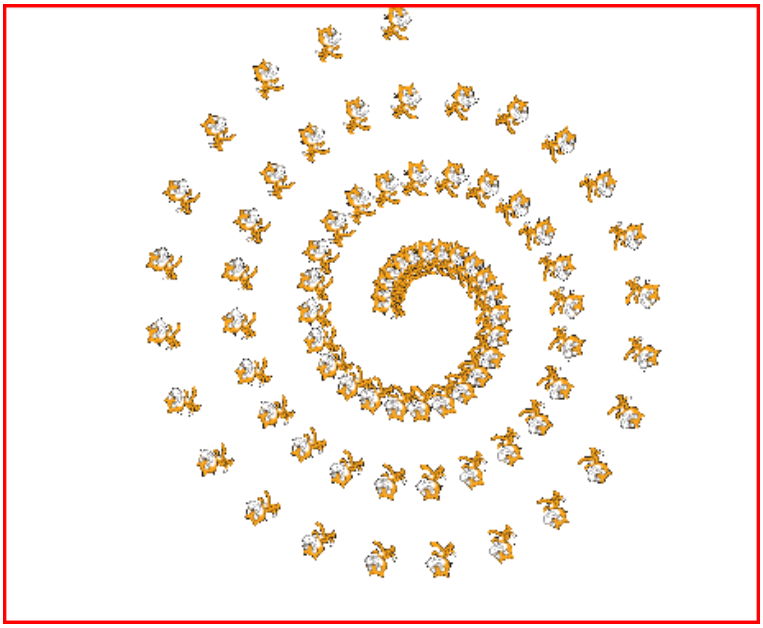


Figura J.12: Scratch – Saída de uma codificação (exemplo6) de movimento com variáveis e repetições

O que poderá ser feito através da codificação apresentada na figura J.13.

Figura J.13: Scratch – Codificação (exemplo6) de movimento com variáveis e repetições

Posteriormente, o professor poderá pedir aos alunos para realizar, uma determinada codificação. Apresentamos como sugestão o comportamento apresentado na figura J.14.

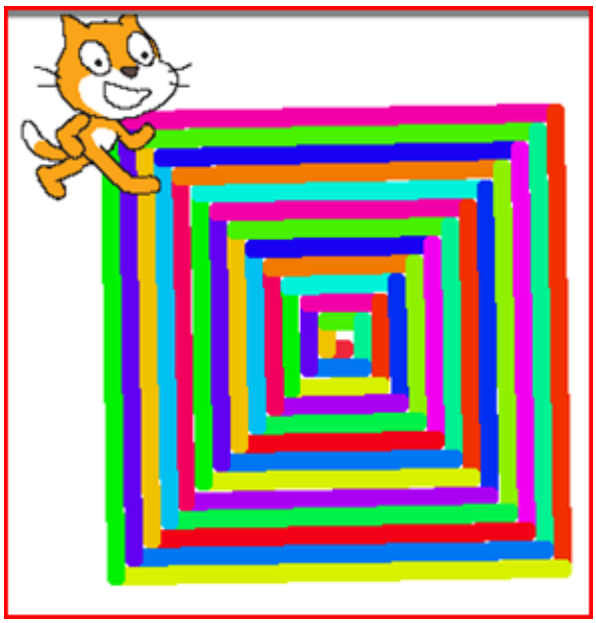


Figura J.14: Scratch – Saída de uma codificação (exemplo7) de movimento com variáveis e repetições

Este comportamento poderá ser feito através da codificação apresentada na figura J.15.

Figura J.15: Scratch – Codificação (exemplo7) de movimento com variáveis e repetições

Se o professor considerar oportuno, mais tarde, mesmo já estando a utilizar uma linguagem de programação específica, poderá recorrer ao Scratch sempre que tiver necessidade de mostrar determinados comportamentos mais gráficos, como os exemplos ilustrados na figura J.16 e na figura J.17.

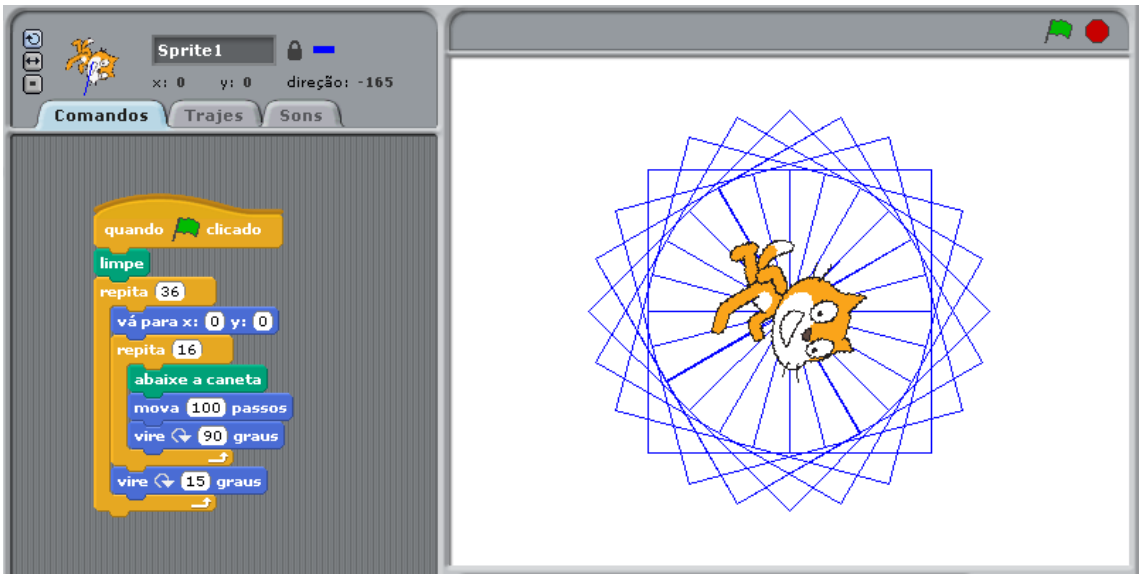


Figura J.16: Scratch – Codificação e saída (exemplo8) de movimento com variáveis e repetições dentro de outras repetições

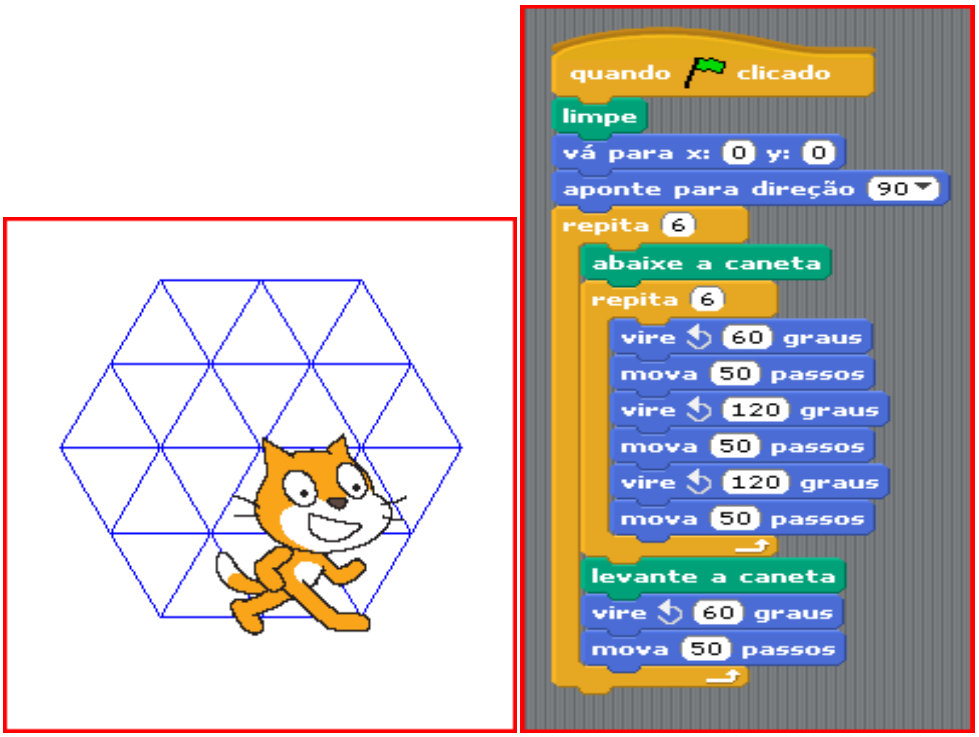


Figura J.17: Scratch – Codificação e saída (exemplo9) de movimento com variáveis e repetições dentro de outras repetições

Anexo K

SICAS

O SICAS (**S**istema **I**nteractivo para **C**onstrução de **A**lgoritmos e sua **S**imulação) (Gomes, 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.

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 ou descrição do problema. Na área de construção/edição 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 K.1 ilustra o cenário de edição/resolução de problemas. A figura K.2, figura K.3, figura K.4, figura K.5, figura K.6, figura K.7, figura K.8 e figura K.9 mostram as janelas correspondentes à simbologia gráfica que representam as principais estruturas necessárias à construção de um algoritmo.

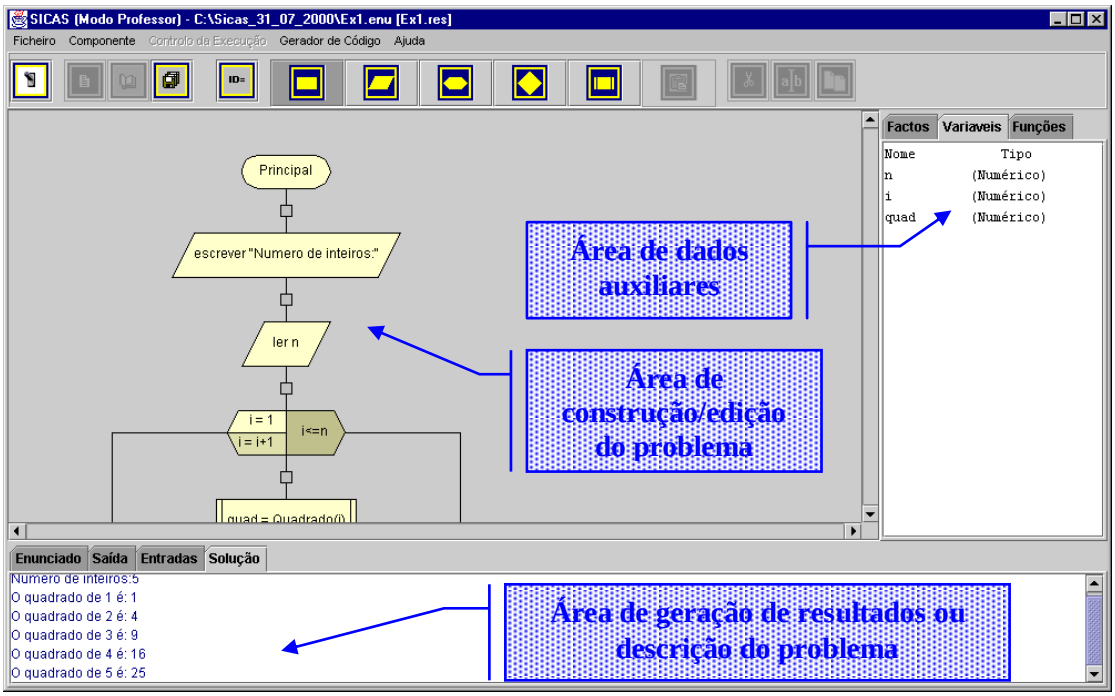


Figura K.1: Interface do SICAS – Construção/edição de algoritmos no SICAS

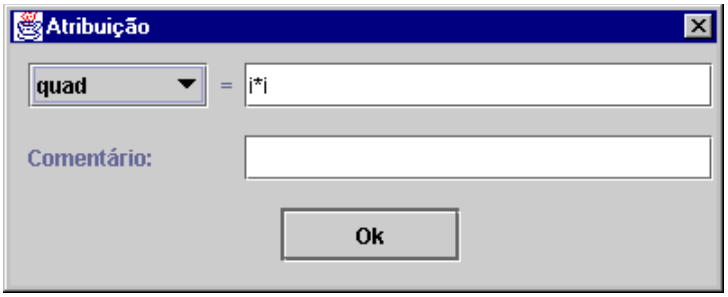


Figura K.2: Interface do SICAS – Diálogo de especificação de uma atribuição

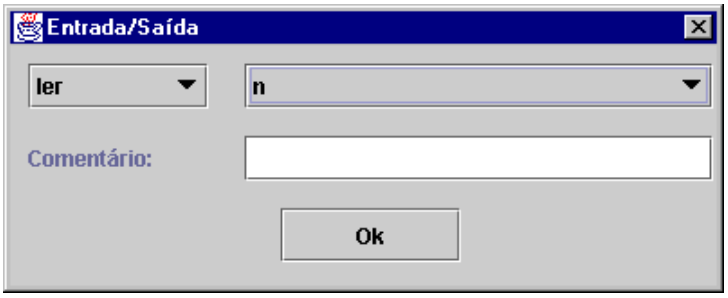


Figura K.3: Interface do SICAS – Diálogo de especificação de uma leitura

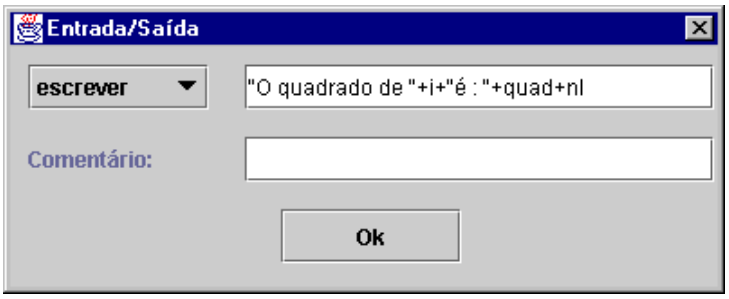


Figura K.4: Interface do SICAS – Diálogo de especificação de uma escrita

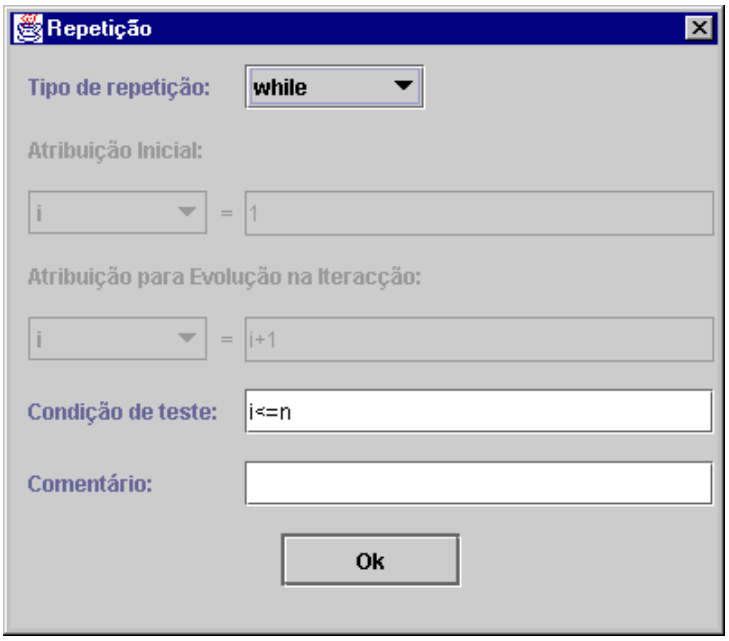


Figura K.5: Interface do SICAS – Diálogo de especificação de uma repetição while

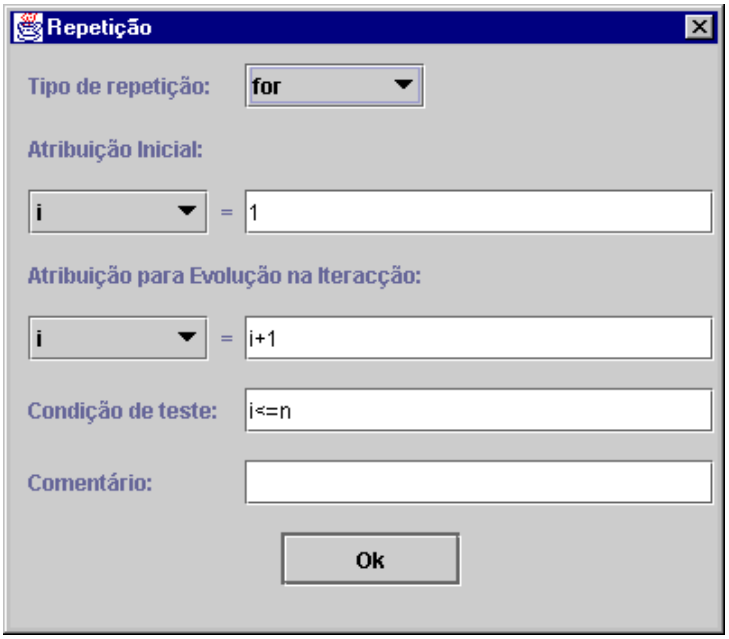


Figura K.6: Interface do SICAS – Diálogo de especificação de uma repetição for

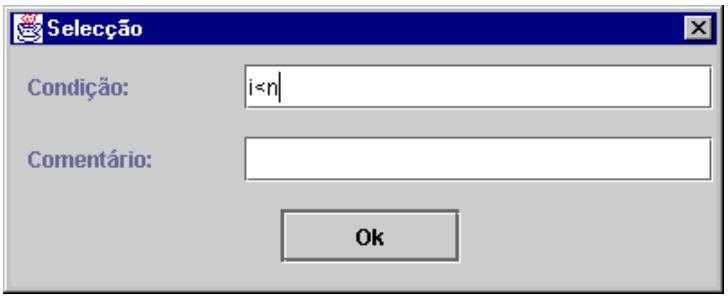


Figura K.7: Interface do SICAS – Diálogo de especificação de uma selecção

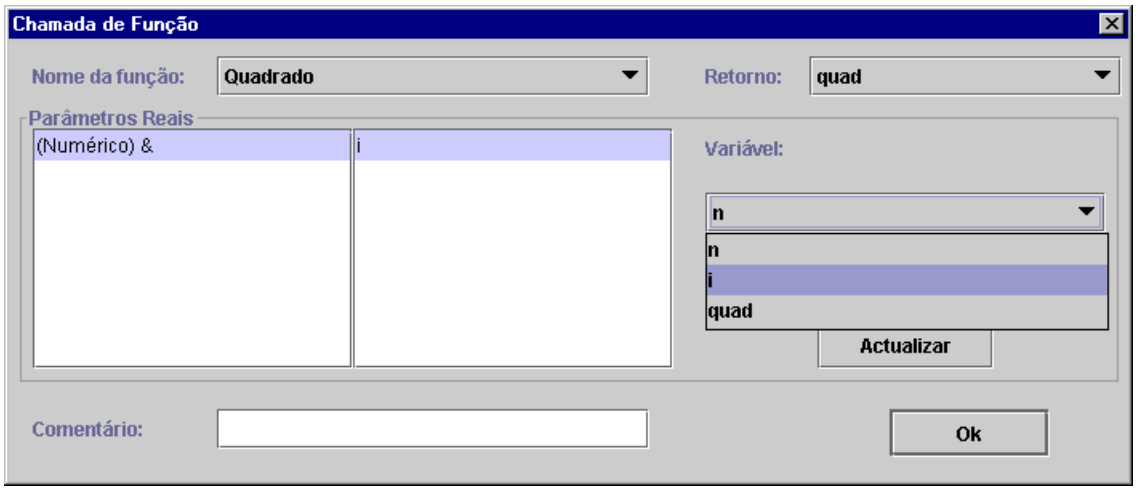


Figura K.8: Interface do SICAS – Diálogo de uma chamada de uma função com passagem de parâmetros por referência

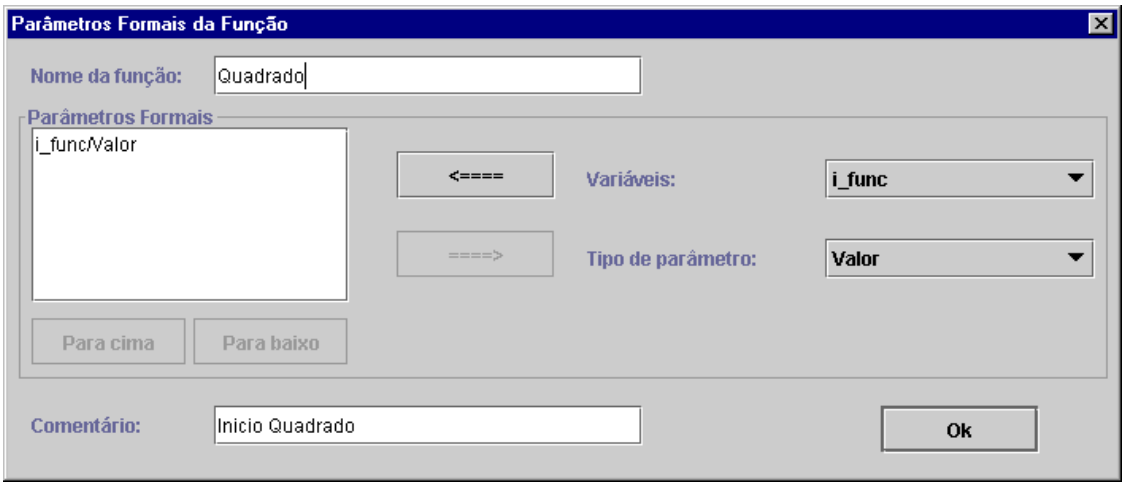


Figura K.9: Interface do SICAS – Diálogo de definição dos parâmetros formais de uma função

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 modo permite um elevado grau de interacção e controlo por parte do utilizador, possibilitando-lhe realizar diversas configurações relativamente ao progresso de uma execução. O aluno também tem 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.

De acordo com a nossa proposta, o SICAS seria adequado para uma melhor aproximação a uma linguagem de programação real antes da utilização dessa mesma linguagem. Assim, seria proposto inicialmente para a demonstração, pelo professor, de conceitos e estruturas de programação, bem como para a realização de problemas de programação muito simples. A título de exemplo, mostramos três problemas básicos, um de carácter puramente sequencial (figura K.10), outro com uma selecção simples (figura K.11) e outro com um ciclo (figura K.12). Em fases posteriores o SICAS poderá também ser utilizado para a realização de exercícios mais complexos, com ciclos dentro de ciclos, manipulação de tabelas ou inclusão de funções.

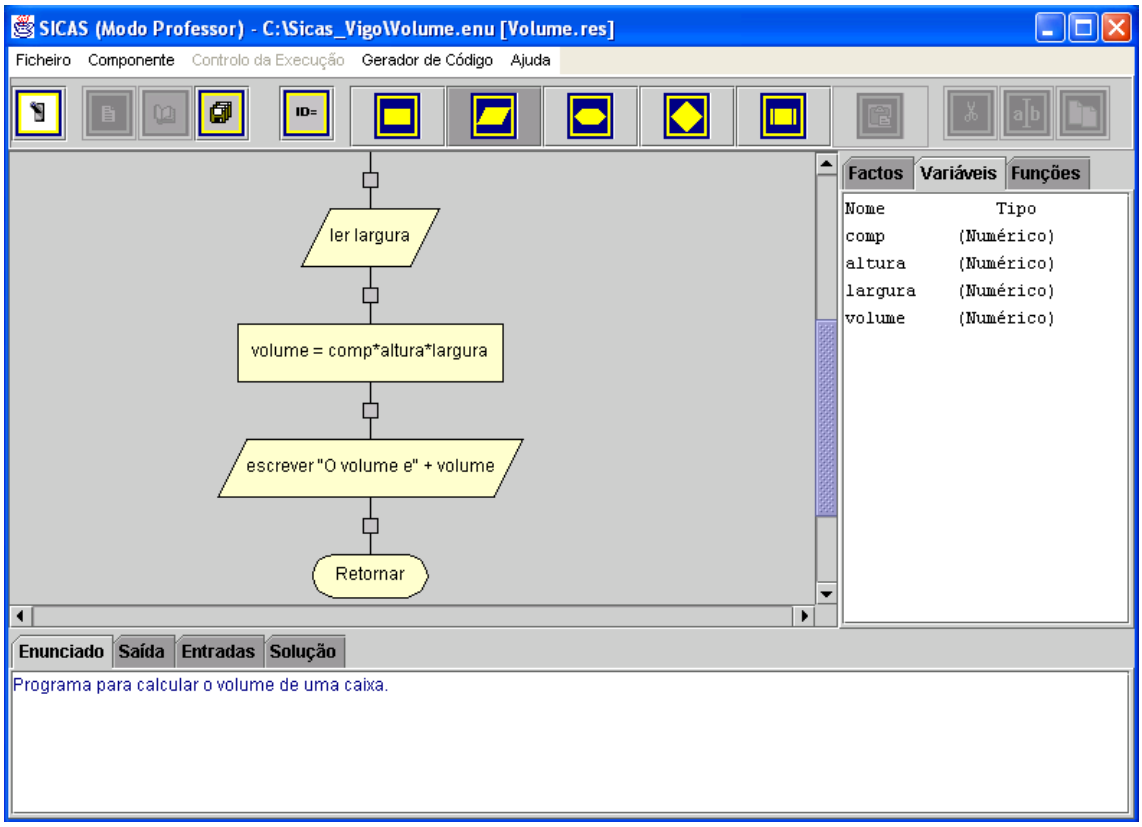


Figura K.10: Interface do SICAS – Exemplo de um problema sequencial

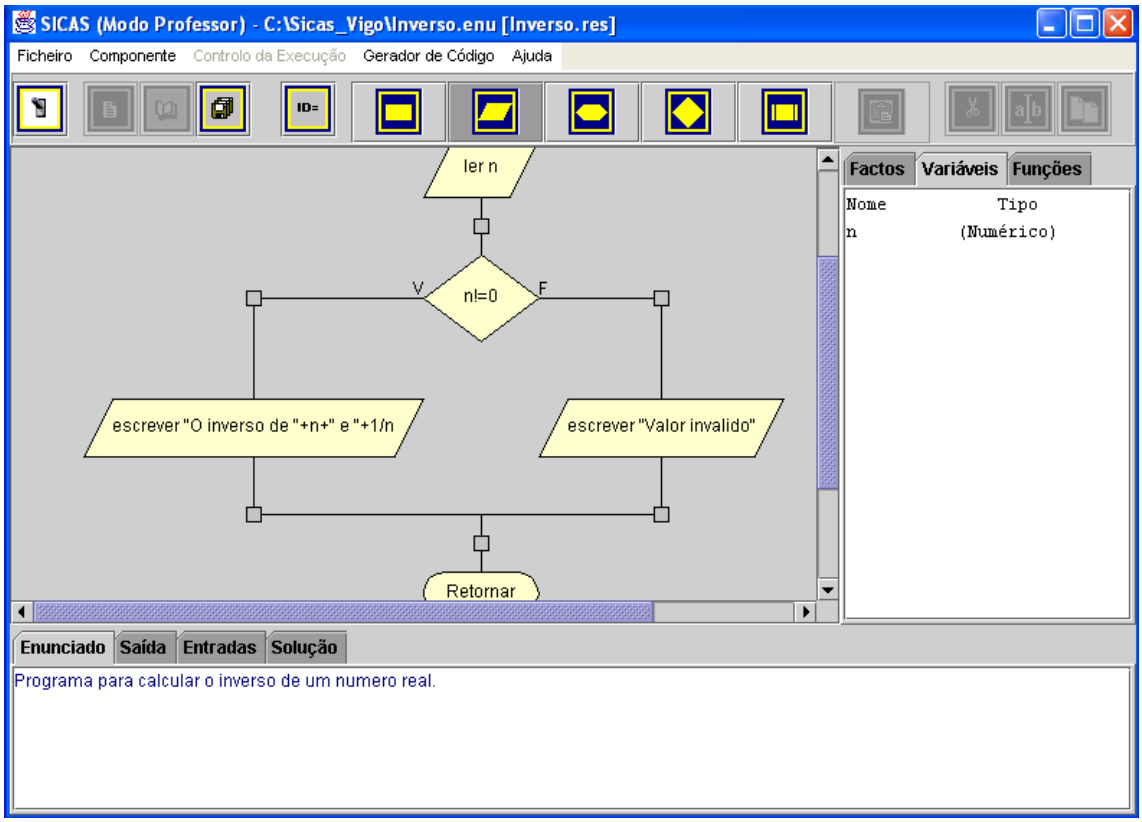


Figura K.11: Interface do SICAS – Exemplo de um problema com uma selecção

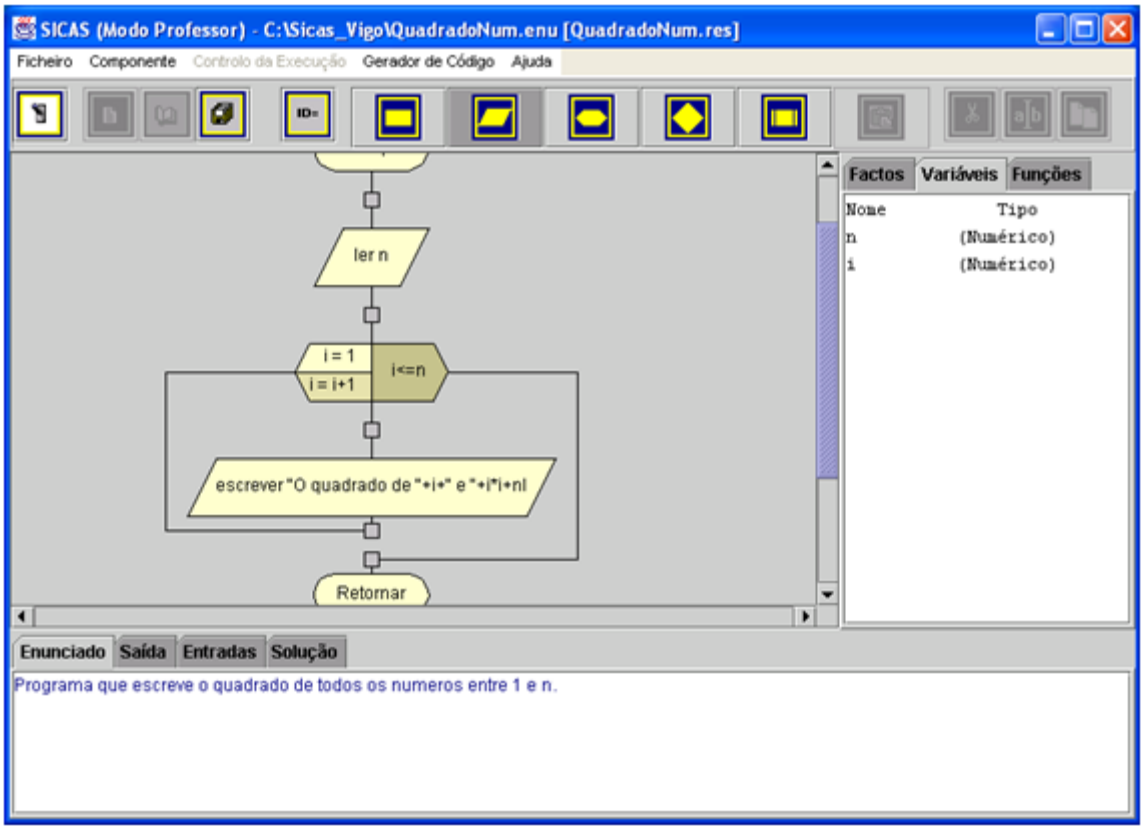


Figura K.12: Interface do SICAS – Exemplo de um problema com um ciclo

Apesar de propormos a utilização do SICAS antes do contacto com a linguagem de programação real a aprender, pensamos que quer o aluno quer o professor poderão a ele recorrer em qualquer momento que acharem adequado. Isto porque 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.

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 apresentar 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.

Anexo L

Actividades Propostas – Componente BIA

De seguida são apresentadas actividades, que podem ser usadas nos diversos níveis da Componente BIA. Apesar das actividades propostas serem baseadas na taxonomia de Bloom, outras abordagens taxonómicas seriam possíveis, cite-se por exemplo, a Taxonomia da Matriz ou a representação em espiral de uma taxonomia. Porém achámos que para cativar os professores para a adopção deste tipo de abordagens seria importante propor aproximações mais fáceis de entender e executar, que tenham dado provas da sua eficácia e sobre as quais existam exemplos mais concretos sobre a sua utilização.

Considerando a taxonomia de Bloom, a tabela L.1 inclui para cada nível, uma pequena explicação e exemplificação do tipo genérico de actividades de programação adequadas.

Porém, o professor deverá ter a flexibilidade e liberdade para a definição de actividades de acordo com as necessidades evidenciadas, sendo o mais importante a existência de actividades de grau crescente de dificuldade. Será ainda necessário um cuidado acrescido aquando da existência de problemas com ênfase matemática, os quais deverão ser acompanhados de explicações e exemplificações necessárias à completa compreensão do conceito envolvido. Caso o professor considere necessário, deverá recomendar o estudo de determinados assuntos matemáticos mais genéricos, como por exemplo, trigonometria.

Nível	Definição	Exemplos de actividades
Conhecimento	O aluno irá recordar, reconhecer ou recuperar informações, ideias e princípios. A base é a memorização.	Reconhecer terminologias, estruturas, métodos ou a sintaxe de várias estruturas de programação.
Compreensão	O aluno traduz, compreende ou interpreta informação com base em conhecimento prévio. Envolve a interiorização do conhecimento.	Ser capaz de interpretar e explicar o comportamento de determinada estrutura de controlo ou pedaço de código, por palavras próprias. Fazer comparações entre códigos. Capacidade para identificar as estruturas de programação de maior utilidade face a determinado conjunto de circunstâncias.
Aplicação	O aluno selecciona, transfere e usa dados e princípios para completar um problema ou tarefa com um mínimo de supervisão.	Reconhecimento de que tipo de elementos são necessários para resolver determinada situação e aplicá-los correctamente. Por exemplo, prever a necessidade de um ciclo dentro de outro ciclo. A necessidade de avaliar uma lista de requisitos e escrever um programa ou conjunto de código que a satisfaça.
Análise	O aluno distingue, classifica e relaciona pressupostos, hipóteses, evidências ou estruturas de uma declaração ou questão. É capaz de explicar a lógica subjacente a um processo.	Capacidade de pegar em ideias conhecidas e descobrir conexões entre elas, resultando numa solução (normalmente criativa) de um problema. A capacidade de interpretar um programa e perceber o que se passa.
Síntese	O aluno cria, integra e combina ideias num produto, plano ou proposta, novos.	Perceber que existem padrões em determinadas resoluções, particionar a informação nos seus elementos constituintes e organizar as partes para formar um todo. Escrever programas com base em especificações de alto nível.
Avaliação	O aluno aprecia, avalia ou critica com base em padrões e critérios específicos.	Explicar porque se concorda ou não com determinada solução. Explicar um pensamento, bem como argumentos contra ou a favor de determinada resolução.

Tabela L.1: Exemplos de actividades de programação para os diferentes níveis da taxonomia de Bloom

Nesta proposta de actividades, é suposto que todos os alunos comecem pelas actividades de nível cognitivo menos exigente (Nível B), e vão progredindo no sentido de atingir as dos níveis seguintes. É claro que, para cada tipo de actividade proposta, os alunos dos diferentes grupos poderão atingir níveis diferentes em cada aula, pelo que o professor deverá estar munido de um conjunto de estratégias para que os alunos atinjam o nível de conhecimentos considerado adequado para um correcto acompanhamento da matéria. Assim, se por exemplo, os alunos mais fracos não atingirem os níveis considerados satisfatórios pelo professor, deverão fazer todos os esforços extra-aulas para os conseguirem na sua plenitude. Para este efeito o professor poderá ainda propor actividades ou estudos suplementares, para que os alunos com mais dificuldades melhor consolidem ou atinjam os conhecimentos pretendidos. De notar ainda, que o professor poderá entender que uma ficha, englobando todos os níveis, poderá ocupar várias aulas, ou pelo contrário, uma ficha poderá não ocupar a totalidade de uma aula, em especial para os alunos mais avançados. Nesta última situação o professor deverá prever a possibilidade de pequenos projectos ou pesquisas mais desafiantes para estes alunos ou envolvê-los em tarefas de ajuda aos alunos com mais dificuldades.

As Ficha Nº1 e a Ficha Nº2 apresentadas não têm uma ordem de aplicação, servem apenas para demonstrar situações e abordagens um pouco diferentes, em que a primeira enfatiza a utilização de ciclos e a segunda utiliza variados conceitos matemáticos.

Nalgumas situações serão dados exemplos de perguntas suplementares que o professor poderá propor de acordo com as necessidades, como garantia de uma correcta passagem ao nível seguinte. Também consideramos que o número de perguntas em cada nível pode variar.

Consideramos também fundamental o professor não esquecer a mensagem fundamental da Taxonomia de Bloom, a utilização de estratégias variadas para testar os alunos em todos os níveis da taxonomia. Porém, quando esta taxonomia foi publicada a principal chamada de atenção era para a ênfase demasiada nos níveis mais baixos da hierarquia. Actualmente e face ao problema específico de programação a chamada de atenção é para o problema inverso: os professores centram-se muito nos últimos níveis da taxonomia ignorando os níveis mais baixos.

Anexo L.1 – Ficha Nº1

****

Nível B

****

Conhecimento

1. Qual é o número mínimo de vezes que um ciclo `while` é executado?
   a) 0.
   b) 1.
   c) Variável.
   d) Uma por cada variável de controlo usada.
2. Qual é o número mínimo de vezes que um ciclo `do…while` é executado?
   a) 0.
   b) 1.
   c) Variável.
   d) Uma por cada variável de controlo usada.
3. Qual é o número mínimo de vezes que um ciclo `for` é executado?
   a) 0.
   b) 1.
   c) Variável.
   d) Uma por cada variável de controlo usada.

Compreensão

1. Relativamente ao código de seguida apresentado, qual das seguintes afirmações é verdadeira?

```
int x;
for(x=0; x<10; x=x+1) {}
```

   a) Na 1ª iteração do ciclo x=0 e na última x=9.

   b) Na 1ª iteração do ciclo x=0 e na última x=10.

   c) Na 1ª iteração do ciclo x=1 e na última x=9.

   d) Na 1ª iteração do ciclo x=1 e na última x=10.

   e) Não existe nenhuma iteração do ciclo porque o x é menor que 10 logo na 1ª iteração.

2. A seguinte codificação

```
int i = 0;
while (i < 10)
    printf("hello");
```

   a) Nunca imprime "hello".

   b) Imprime "hello" 9 vezes.

   c) Imprime "hello" 10 vezes.

   d) Imprime "hello" infinitamente.

Compreensão – Perguntas Suplementares

1. Qual das seguintes codificações imprime um conjunto de n "*", um em cada linha? Por exemplo, para n=5, apresentará a seguinte saída:

```
*
*
*
*
*
```

   a)

```
void linha(int n){
    int i;
    for(i=1;i<=n;i++)
        printf("*");
    printf("\n");
}
```

b)

```
void linha(int n){
    int i;
    for(i=1;i<=n;i++)
        printf("\n");
    printf("*");
}
```

c)

```
void linha(int n){
    int i;
    for(i=1;i<=n;i++)
        printf("*\n");
}
```

d) Nenhuma das anteriores.

2. Dada a seguinte função, indique o resultado final, para n=6:

```
void linha(int n){
    int i;
    for(i=n;i>=0;i=i-1)
        printf("%d",i);
    printf("%d",n-i);
}
```

a) 65432107.

b) 65432106.

c) 01234560.

d) 60514233241506.

e) 06152433425160.

f) Nenhuma das anteriores.

****

Nível I

****

Aplicação

1. Implemente o código necessário de forma a escrever n linhas, colocando em cada linha os dígitos até n-1. Por, exemplo, para n=5, produzirá o seguinte resultado.

```
1234
1234
1234
1234
1234
```

2. Escreva o código necessário de forma a produzir o seguinte resultado para n=5.

```
1
12
123
1234
12345
```

Aplicação – Perguntas Suplementares

1. Escreva o código necessário de forma a produzir o seguinte resultado para n=5.

```
5
54
543
5432
54321
```

2. Escreva o código necessário de forma a produzir o seguinte resultado para n=5.

```
12345
1234
123
12
1
```

Análise

1. Realize o código necessário de forma a produzir o seguinte resultado para n=5.

```
54321
 5432
  543
   54
    5
```

2. Realize o código necessário de forma a produzir o seguinte resultado para n=5.

```
    1
   21
  321
 4321
54321
```

Análise – Perguntas Suplementares

1. Realize o código necessário de forma a produzir o seguinte resultado para n=5.

```
54321
 4321
  321
   21
    1
```

2. Realize o código necessário de forma a produzir o seguinte resultado para n=5.

```
    5
   54
  543
 5432
54321
```

****

Nível A

****

Síntese

1. Suponha que a seguinte função permite obter a figura de seguida apresentada:

```
void linha(int n){
    int i,j,k;
    for(i=1;i<=n;i++){
        for(j=1;j<=n-i;j++)
            printf(" ");
        for(k=1;k<=i;k++)
            printf("*");
        printf("\n");
    }
}
```

```
    *
   **
  ***
 ****
*****
```

E que a seguinte função permite obter a figura de seguida apresentada:

```
void linha(int n){
    int i,j;
    for(i=1;i<=n;i++){
        for(j=1;j<=i;j++)
            printf("*");
        printf("\n");
    }
}
```

```
*
**
***
****
*****
```

Realize o código para produzir o seguinte resultado.

```
    *
   ***
  *****
 *******
*********
```

2. Supondo que, para além das funções anteriores tem ainda as seguintes funções que permitem obter as figuras de seguida apresentadas:

```
void linha(int n){
    int i,j,k;
    for(i=1;i<=n;i++){
        for(j=1;j<i;j++)
            printf(" ");
        for(k=1;k<=n-i+1;k++)
            printf("*");
        printf("\n");
    }
}
```

```
*****
 ****
  ***
   **
    *
```

```
void linha(int n){
    int i,j;
    for(i=1;i<=n;i++){
        for(j=1;j<=n-i+1;j++)
            printf("*",j);
        printf("\n");
    }
}
```

```
*****
****
***
**
*
```

Realize o código para produzir o seguinte resultado.

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

Síntese – Perguntas Suplementares

1. Realize o código para produzir o seguinte resultado.

```
    1
   222
  33333
 4444444
555555555
 4444444
  33333
   222
    1
```

2. Realize o código para produzir o seguinte resultado.

```
    1
   121
  12321
 1234321
123454321
 1234321
  12321
   121
    1
```

Avaliação

1. Relativamente aos problemas anteriores, quais as alterações que teria de realizar se pretendesse obter um losango com o seguinte aspecto?

```
    *
   * *
  *   *
 *     *
*       *
 *     *
  *   *
   * *
    *
```

2. Avalie o grau de similaridade entre a resolução do problema anterior e de outro que gere a seguinte figura.

```
    *
   *^*
  *^*^*
 *^*^*^*
*^*^*^*^*
 *^*^*^*
  *^*^*
   *^*
    *
```

Alternativamente o professor poderia propor para actividades deste último nível, que os alunos analisassem as soluções dos seus colegas, face aos problemas propostos, para que as discutissem em termos de aspectos de mais alto nível como a legibilidade, clareza de estruturação e facilidade de interpretação do código, erros encontrados ou outras questões de optimização de código.

Anexo L.2 – Ficha Nº2

****

Nível B

****

Conhecimento

1. Dado o seguinte extracto de código, responda às seguintes questões:

```
1:  void main() {
2:      int n1,n2,i;

3:      n1=3;
4:      n2=30;
5:      for(i=n1;i<=n2;i=i+1){
6:          if (n2%i==0)
7:              printf("%d\n",i);
8:      }
9:  }
```

   a) Existe algum ciclo? Se sim, indique qual a linha em que é iniciado e terminado?

   b) Existe alguma instrução de selecção? Se sim, indique qual a linha em que é iniciada e terminada?

Compreensão

1. Responda às seguintes questões:

   a) Supondo i=101 qual o resultado da seguinte expressão `i%2`?

   b) O que se testa na seguinte expressão `if (n2%i==0)`?

   c) Explique o que faz a instrução `for(i=n1;i<=n2;i=i+1).`

   d) Explique, de forma sintetizada, o que faz o pedaço de código acima.

Compreensão – Perguntas Suplementares

1. Assuma que as linhas 3 e 4 são substituídas pelo seguinte código:

```
3:  n1:=30;
4:  n2:=3;
```

   Qual o resultado desta modificação no código acima?

2. Introduza as alterações necessárias no programa dado inicialmente, considerando os dados da pergunta anterior, de forma a que o comportamento do programa seja o mesmo do fornecido inicialmente.

****

Nível I

****

Aplicação

1. Altere o programa anterior de forma a que apresente no ecrã os divisores comuns entre dois números.

2. Altere o programa anterior para que seja apenas mostrado o mínimo múltiplo comum entre n1 e n2.

Análise

1. Analise o programa seguinte dizendo qual o seu objectivo.

```
void main() {
    int n1,n2,i,j, maior, menor;
    n1=3;
    n2=30;
    if (n1>n2){
        maior=n1;
        menor=n2;
    }else{
        maior=n2;
        menor=n1;
    }
    for(i=menor; i>=1; i=i-1) {
        if (menor%i==0 && maior%i==0)
            break;
    }
    printf("%d",i);
}
```

2. Sabendo que o seguinte programa tem o mesmo objectivo que o anterior, analise-os comparando a sua eficiência.

```
int main() {
    int m=3, n=30, r;
    while ((r=m%n)!=0) {
        m=n;
        n=r;
    }
    printf("O resultado e %d\n", n);
    return 0;
}
```

Análise – Perguntas Suplementares

1. Realize um programa que permita verificar se um número é ou não primo, imprimindo uma mensagem informativa desse facto.

   Número primo

   Um número primo é um número inteiro p>1 que não tem divisores inteiros a não ser o número 1 e p. Se p não é um número primo diz-se que é um número composto.
   Exemplo: Os únicos divisores do número 7 são o número 1 e o número 7, enquanto o número 24 tem por divisores os números 1, 2, 3, 4, 6, 8, 12 e 24, logo 7 é um número primo e 24 não é um número primo.

2. Dada a seguinte função cujo objectivo é determinar se um número é ou não primo, deduza qual a condição while. Justifique a sua opção.

```
void NumPrimo(int i) {
    int j, flag=1;

    if i>4{
        j=2;
        while ( COMPLETAR ) {
            if (i%j==0)
                flag=0;
            j++;
        }
    }
    if (flag)
        printf("O número %d é primo\n",i);
    else
        printf("O número %d não é primo\n",i);
}
```

****

Nível A

****

Síntese

1. Realize um programa que verifique se um número n é perfeito.

   Número perfeito

   Dizemos que um número inteiro é um número perfeito se a soma dos seus divisores, incluindo 1, mas não incluindo o próprio número, é igual ao número. Por exemplo, 6 é um número perfeito, pois 6 = 1 + 2 + 3.

2. Realize um programa que permita decompor um dado número nos seus factores primos, indicando adicionalmente o grau de divisibilidade de cada um. O resultado deverá seguir o seguinte exemplo:

   Decomposição de 6 em Factores Primos:
   6: 2(1) 3(1)
   Decomposição de 36 em Factores Primos:
   36: 2(2) 3(2)
   Decomposição de 37 em Factores Primos:
   37: É um Numero Primo
   Como sugestão poderá usar o seguinte algoritmo.

   Algoritmo da Factorização em números primos
   Este algoritmo pode ser aplicado a qualquer número natural p e permite-nos verificar se p é um número primo ou composto e, neste caso, quais são os seus factores primos. Para o implementar procede-se da seguinte forma: tenta-se, sucessivamente, dividir p por cada número primo n = 2, 3, 7, … até ao maior numero primo não superior a $\sqrt{p}$. Se o resto da divisão for sempre diferente de zero então p é um número primo. Se, digamos, p0, dividir p, p é composto e escreve-se p=p0p1, com p1<p0, repete-se o mesmo processo com p1, começando sempre do último número primo que permitiu obter resto zero. Desta forma consegue-se obter a factorização completa de p em números primos.
   Por exemplo, a 24 corresponde a factorização em números primos 2^3 x 3. Uma vez que
   $24 \div 2 = 12 \qquad 24 = 2 \times 12$
   12 não é um número primo por isso voltamos a dividir o quociente, 12, por 2
   $12 \div 2 = 6 \qquad 24 = 2 \times 2 \times 6$
   6 não é um número primo por isso voltamos a dividir o quociente, 6, por 2
   $6 \div 2 = 3 \qquad 24 = 2 \times 2 \times 2 \times 3 = 2^3 \times 3$
   3 é um número primo podemos parar o processo.

Avaliação

1. Os gregos descobriram que todo o número perfeito é da forma $n=2^{m-1}x(2^m-1)$, onde m é um inteiro com m>=2 e 2^m-1 é primo. Implemente um programa que teste se um número é perfeito, de acordo com este processo, avaliando esta solução, em termos de eficiência, relativamente ao algoritmo proposto em pergunta anterior (um número inteiro é um número perfeito se a soma de seus divisores, incluindo 1, mas não incluindo o próprio número, é igual ao próprio número).

2. De acordo com o ponto anterior, um número da forma 2^m-1 é chamado de número de Mersenne. Um número primo dessa forma é chamado de número primo de Mersenne. Implemente um programa para encontrar *Números primos de Mersenne* até um determinado limite (ex: 700). Para a implementação, apresente duas formas para testar a primalidade de um número, recorrendo, por exemplo, ao algoritmo desenvolvido por si em pergunta anterior e ao teste de Lucas-Lehmer.

   O teste de Lucas – Lehmer refere que seja $S_0=4$; $S_1=4^2-2$;… $S_{k+1}=S_k^2-2$; dado p>2, 2^p-1 é primo se e só se S_{p-2} é múltiplo de 2^p-1.
   Avalie a eficiência de cada solução apresentada.

Avaliação – Perguntas Suplementares

1. O teorema de Fermat, enunciado em 1640, diz que para testar se um número inteiro positivo é primo é necessário calcular potências em que quer a base quer o expoente são números inteiros positivos. Ou seja, dado um número inteiro maior do que 2, tentar encontrar uma prova de que o número é composto (ser composto é não ser primo). Essa prova é um número inteiro, chamado "testemunha". Considere que **N** e **W** são números inteiros, com **N > 2** e **W >= 2**. Diz-se que **W** é testemunha de **N** (ou, mais explicitamente, que **W** é testemunha de que **N** é composto) se o resto da divisão inteira de $\mathbf{W^{N-1}}$ por **N** for diferente de 1. Por exemplo, 2 é testemunha de 4, porque o resto da divisão inteira de 2^3 por 4 é zero. Aplicando o teste de Fermat, conclui-se que um número inteiro N (com N > 2) é primo se nenhum número no conjunto {2, 3, ..., N-1} for testemunha de N. Escreva uma função que, dado um número inteiro N > 2 determina se N é primo de acordo com o teste de Fermat.

2. Um número composto N tal que 2 não é uma testemunha de N diz-se um pseudoprimo binário. Assim, segundo o pequeno teorema de Fermat – se um número p da forma a^{p-1}-1 é divisível por p tem grande probabilidade de ser primo. Por exemplo, 341 é um número composto (341 = 11 * 31) e 2340 mod 341 = 1. Escreva uma função para verificar se um número inteiro maior ou igual a 2 é um pseudoprimo binário. Existem outros algoritmos probabilísticos, que permitem determinar se um número é um psesudo-primo, como o algoritmo de Miller-Rabin. O algoritmo de Miller-Rabin refere que: seja N o inteiro sobre o qual desejamos testar a primalidade e k um factor de controlo de precisão. Se N é composto o algoritmo de Miller-Rabin refere que N é primo com probabilidade 4^{-k}. Se N é primo o algoritmo acerta sempre. Observe que o factor k representa um compromisso entre precisão e desempenho. Implemente estes dois algoritmos, avaliando a sua eficácia computacional.

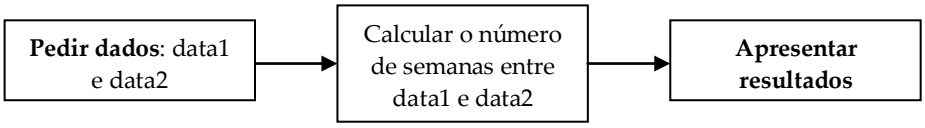
Anexo M

Actividades Propostas – Componente Refina

Desenvolva um programa que peça ao utilizador duas datas (no formato dd/mm/aaaa) e calcule e mostre o número de semanas entre essas duas datas. Caso o número de semanas não seja exacto, deverá ainda mostrar o número de dias adicionais.

T1 – Programa principal estruturado em funções

Estratégia



Para calcular o número de semanas entre duas datas há que saber o número de dias entre essas mesmas datas.

Simplificação

- Assumir a existência de uma função `PedeData` que pede uma data ao utilizador.
- Assumir a existência de uma função `CalculaDiasEntreDatas` que calcula o número de dias entre duas datas.

Resolução

```
void main(void)
{
    int d1,m1,a1;
    int d2,m2,a2;
    int nrdias,nrsemanas,nrdiasadicionais;

    printf("Introduza a primeira data.\n");
    PedeData(&d1,&m1,&a1);
    printf("Introduza a segunda data.\n");
    PedeData(&d2,&m2,&a2);
```

```
    nrdias = CalculaDiasEntreDatas(d1,m1,a1,d2,m2,a2);
    nrsemanas = nrdias / 7;
    nrdiasadicionais = nrdias % 7;

    printf("Entre a data %d/%d/%d e a data %d/%d/%d existem %d semanas
e %d dias.\n\n", d1,m1,a1, d2,m2,a2, nrsemanas, nrdiasadicionais);
}
```

T2 – Implementação das funções

T2.1 – Pedir datas e armazená-las em variáveis

Simplificação

– Sem validação de dados (assume-se que a data introduzida é válida).

Resolução

```
void PedeData(int *d,int *m, int *a){
    printf("Introduza a data no formato dd/mm/aaaa\n");
    scanf("%2d/%2d/%4d", d,m,a);
}
```

T2.2 – Calcular o nº de dias entre duas datas (d1/m1/a1 e d2/m2/a2)

Estratégia

Se datas pertencem ao mesmo ano e mês
NumeroDias=d2-d1
Senão
Se datas pertencem ao mesmo ano
Calcular dias de data1 até ao final do mês
Acumular os dias de cada mês entre m1+1 e m2-1
Acumular d2
Senão
Calcular dias de data1 até ao final do mês
Acumular dias dos meses até ao final do ano
Acumular dias dos anos de a1+1 a a2-1
Acumular dias dos meses do ano a2 até m2-1
Acumular d2

Simplificação

– Assumir a existência de uma função que calcula os dias de cada mês: `int DiasMes(int mes, int ano).`

– Assumir a existência de uma função que verifica se um ano é Bissexto: `AnoBissexto(int ano).`

– Sem validação de dados (assume-se que a 2ª data é superior à 1ª).

Resolução

```
int CalculaDiasEntreDatas(int d1,int m1,int a1,int d2,int m2, int a2)
{
    int i;
    int TotalDias=0;

    if (a1==a2 && m1==m2)
           TotalDias=d2-d1;
    else if (a1==a2){
           TotalDias=DiasMes(m1,a1) - d1;
           for(i=m1+1;i<m2;i++)
               TotalDias+=DiasMes(i,a1);
           TotalDias+=d2;
    } else {
        TotalDias=DiasMes(m1,a1) - d1;
        for(i=m1+1;i<=12;i++)
            TotalDias+=DiasMes(i,a1);
        for(i=a1+1;i<a2;i++)
            TotalDias+=(AnoBissexto(i) ? 366 : 365);
        for(i=1;i<m2;i++)
            TotalDias+=DiasMes(i,a2);
        TotalDias+=d2;
    }
    return TotalDias;
}
```

T2.2.1 – Determinar o número de dias de cada mês

Resolução

Versão 1

```
int DiasMes(int m, int ano){
    int dias;

    if (m==1 || m==3 || m==5 || m==7 || m==8 || m==10 || m==12)
        dias=31;
    else if (m==4 || m==6 || m==9 || m==11)
        dias=30;
    else
        if (AnoBissexto(m))
            dias=29;
        else
            dias=28;
    return dias;
}
```

Versão 2

```
int DiasMes(int mes, int ano)
{
    int tabDiasMes[12]={31,28,31,30,31,30,31,31,30,31,30,31};
    if (mes==2 && AnoBissexto(ano))
        return 29;

    return tabDiasMes[mes-1];
}
```

T2.2.2 – Verificar se um ano é Bissexto

Resolução

```
int AnoBissexto(int ano)
{
    return (ano%400==0) || ((ano%4==0) && (ano%100!=0));
}
```

T3 – Validação de dados

T3.1 – Verificar se a data introduzida é válida

Resolução

```
void PedeData(int *d,int *m, int *a){
    do{
        printf("Introduza a data no formato dd/mm/aaaa\n");
        if (scanf("%2d/%2d/%4d", d,m,a)!=3)
            continue;
    }while(!valida(*d,*m,*a));
}
```

Versão 1 – função valida

```
int valida(int d, int m, int a){
    if ((m==1 || m==3 || m==5 || m==7 || m==8 || m==10 || m==12) &&
d>31)
        return 0;
    else if ((m==4 || m==6 || m==9 || m==11) && d>30)
        return 0;
    else if (AnoBissexto(a) && m==2 && d>29)
        return 0;
    else if (!AnoBissexto(a) && m==2 && d>28)
        return 0;
    return 1;
}
```

Versão 2 – função valida

```
int valida(int d, int m, int a){
    return (a > 0 && m > 0 && m < 12 && d > 0 && d <= DiasMes(m,a));
}
```

T3.2 – Verificar se a segunda data é superior à primeira

Resolução

```
void main(void)
{
    ...

    do{
        printf("Introduza a primeira data.\n");
        PedeData(&d1,&m1,&a1);
        printf("Introduza a segunda data.\n");
        PedeData(&d2,&m2,&a2);
    }while(!verifica_2_maior_1(d1,m1,a1,d2,m2,a2));
    ...
}
```

Versão 1 – função verifica_2_maior_1

```
int verifica_2_maior_1(  int d1, int m1, int a1,
```

```
                          int d2, int m2, int a2 )
{
    if (a2<a1)
        return 0;
    if (a2>a1)
        return 1;
    if (m2<m1)
        return 0;
    if (m2>m1)
        return 1;
    return (d2>=d1);
}
```

Versão 2 – função verifica_2_maior_1

```
int verifica_2_maior_1(  int d1, int m1, int a1,
                          int d2, int m2, int a2 )
{
    return (a2<a1 || (a2==a1 && m2<m1) ||
            (a2==a1 && m2==m1 && d2<d1) );
}
```

Versão 3 – função verifica_2_maior_1

Estratégia

Para verificar se a 2ª data é superior à 1ª, basta utilizar a seguinte fórmula:

d1 + m1 * 32 + a1 * 512 <= d2 + m2 * 32 + a2 * 512

Resolução

```
int verifica_2_maior_1(  int d1, int m1, int a1,
                          int d2, int m2, int a2 )
{
    return (d1 + m1 * 32 + a1 * 512 <= d2 + m2 * 32 + a2 * 512);
}
```

A totalidade deste código é de seguida apresentado.

```
#include <stdio.h>

int AnoBissexto(int ano)
{
    return (ano%400==0) || ((ano%4==0) && (ano%100!=0));
}

int DiasMes(int mes, int ano)
{
    int tabDiasMes[12]={31,28,31,30,31,30,31,31,30,31,30,31};

    if (mes==2 && AnoBissexto(ano))
        return 29;
    return tabDiasMes[mes-1];
}

int valida(int d, int m, int a){
    return (a > 0 && m > 0 && m < 12 && d > 0 && d <= DiasMes(m,a));
}

void PedeData(int *d,int *m, int *a){
    do{
        printf("Introduza a data no formato dd/mm/aaaa\n");
```

```
        if (scanf("%2d/%2d/%4d",d,m,a)!=3)
            continue;
    }while(!valida(*d,*m,*a));
}

int verifica_2_maior_1(  int d1, int m1, int a1,
                         int d2, int m2, int a2)
{
    return (d1 + m1 * 32 + a1 * 512 <= d2 + m2 * 32 + a2 * 512);
}

int CalculaDiasEntreDatas(int d1,int m1,int a1,int d2,int m2, int a2)
{
    int i;
    int TotalDias=0;

    if (a1==a2 && m1==m2)
        TotalDias = d2 - d1;
    else if (a1==a2){
        TotalDias = DiasMes(m1,a1) - d1;
        for(i=m1+1;i<m2;i++)
            TotalDias += DiasMes(i,a1);
        TotalDias += d2;
    } else {
        TotalDias = DiasMes(m1,a1) - d1;
        for(i=m1+1;i<=12;i++)
            TotalDias += DiasMes(i,a1);
        for(i=a1+1;i<a2;i++)
            TotalDias += (AnoBissexto(i) ? 366 : 365);
        for(i=1;i<m2;i++)
            TotalDias += DiasMes(i,a2);
        TotalDias += d2;
    }
    return TotalDias;
}

void main(void)
{
    int d1,m1,a1;
    int d2,m2,a2;
    int nrdias,nrsemanas,nrdiasadicionais;

    do{
        printf("Introduza a primeira data.\n");
        PedeData(&d1,&m1,&a1);
        printf("Introduza a segunda data.\n");
        PedeData(&d2,&m2,&a2);
    }while(!verifica_2_maior_1(d1,m1,a1,d2,m2,a2));

    nrdias = CalculaDiasEntreDatas(d1,m1,a1,d2,m2,a2);
    nrsemanas = nrdias / 7;
    nrdiasadicionais = nrdias % 7;

    printf("Entre a data %d/%d/%d e a data %d/%d/%d existem %d semanas
e %d dias.\n\n", d1,m1,a1, d2,m2,a2, nrsemanas, nrdiasadicionais);
}
```