Utilizando Programação Funcional em Disciplinas Introdutórias de Computação

Thais Helena Chaves de Castro[1,2], Alberto Nogueira de Castro Júnior[2], Crediné Silva de Menezes[1], Maria Cláudia Silva Boeres[1], Maria Christina Pedrosa Valli Rauber[1]

[1]Departamento de Informática – Universidade Federal do Espírito Santo (UFES)
Av. Fernando Ferrari s/n CT VII – 29.060-970 – Vitória – ES – Brasil

[2]Departamento de Ciência da Computação – Universidade Federal do Amazonas (UA)
Av. Rodrigo O. J. Ramos, 3000 – 69.077-000 – Manaus – AM – Brasil

{thaish,credine}@inf.ufes.br, albertoc@dcc.fua.br

Abstract. *This paper describes an experiment which has been developed for the last eight years, with the use of functional paradigm to teach programming. Firstly, an overview on the state-of-the-art in the programming teaching task, is presented. Next, it is discussed how challenging the task of teaching programming to first-year undergraduate students is, and the different approaches to do that. Finally, a case study involving introductory programming courses is presented and discussed.*

Resumo. *Este artigo descreve um experimento desenvolvido ao longo dos últimos oito anos, com o uso do paradigma funcional no ensino introdutório de programação. Inicialmente é apresentado um levantamento do estado da arte no ensino de programação. A seguir, discute-se como é desafiadora a tarefa de ensinar programação nos cursos introdutórios de graduação, e as diferentes abordagens utilizadas com tal finalidade. Por fim, um estudo de caso envolvendo cursos introdutórios de programação que utilizam o paradigma funcional é apresentado e discutido.*

Palavras-chave. *ensino de programação, programação funcional, metodologia de ensino de programação.*

1. Introdução

O ensino de programação é desafiador porque a própria tarefa de programar não é fácil. Devido a isso, as universidades têm discutido com freqüência seus currículos de Ciência da Computação e Engenharia da Computação, em busca de alternativas para diminuir o índice de abandono desses cursos e as dificuldades encontradas por estudantes iniciantes.

Dentre as várias abordagens discutidas no momento, a mais tradicional é a de um curso baseado na programação imperativa. Essa abordagem tem trazido sérios problemas para quem nunca programou, pois exige que se repense a forma de resolver um problema como uma "receita de bolo", o que não é o que fazemos no dia-a-dia. Uma abordagem que vem ganhando força nos últimos anos é a que se baseia em programação

funcional, pois, dizem seus defensores, ela de início facilita um "nivelamento" entre os estudantes porque poucos tiveram a experiência de já ter programado usando tal formalismo, além de utilizar um raciocínio matemático, com o qual os estudantes calouros já estão acostumados. Advoga-se ainda o fato de que para uma primeira aproximação com a área de programação de computadores, o uso de linguagens procedurais exige um esforço cognitivo muito grande, tendo em vista que além de se criar o hábito de resolver problemas com o computador, o aluno precisa aprender um novo modelo computacional. O uso de programação funcional reforça a utilização de um ferramental fundamental para a modelagem de problemas e concentra os esforços em uma abordagem mais rigorosa, facilitando inclusive a discussão introdutória de análise de programas.

Na Seção 2, apresentamos uma visão geral sobre a tarefa de aprender a programar, passando depois a descrever o estado da arte no ensino de programação para iniciantes (Seção 3), seguido de um relato de experiências com o paradigma funcional (Seção 4). Em seguida, discutimos como estudo de caso, um experimento de uma universidade que utiliza linguagem funcional para cursos introdutórios em computação (Seção 5).

2. A Aprendizagem de Programação

A aprendizagem de conceitos e métodos para a construção de programas de computador não é trivial, pois requer o uso de habilidades de alto nível e boas doses de raciocínio abstrato. Em [DIJKSTRA 1982] é ressaltado que programar, mais que qualquer outra atividade, envolve a habilidade de raciocinar. Mas programar é também uma tarefa de engenharia, uma vez que trata da produção de artefatos que devem satisfazer requisitos de qualidade e serem passíveis de verificação.

Contudo, em [McKEOWN e FARRELL 1999] é ressaltado que nos cursos introdutórios raramente os estudantes aprendem técnicas de solução de problemas. Nesses cursos, o que ocorre freqüentemente é que os estudantes encontram uma dificuldade muito grande em aplicar suas habilidades prévias. Isso acaba tornando-se uma fonte de medo e frustração, incentivando a evasão.

Nesse contexto, dois problemas se destacam: Por um lado, pouca ênfase é dada às habilidades para entender e solucionar problemas nesses cursos. Por outro, o estudante ainda tem de transformar cada solução num conjunto de comandos executável por computador. Essa transformação é possivelmente a principal dificuldade de todo iniciante em programação. Em primeiro lugar, estudantes tendem a pensar na máquina de modo antropomórfico, isto é, dotando-a de qualidades humanas, assumindo que ela poderia entender comandos incompletos ou errôneos. A necessidade de descrever comandos com rigorosa precisão, torna-se assim uma característica muito artificial e até mesmo *antipática* para o estudante.

São comuns também: as dificuldades no entendimento do significado de um comando enquanto componente de uma solução; a sintaxe do comando numa certa linguagem de programação; e o resultado da execução daquele comando pela máquina. Essas dificuldades são encontradas durante a elaboração de um programa. Além de ter de encontrar comandos para a solução de um problema, o estudante deverá ordená-los

corretamente para criar um programa e, finalmente, verificar e ajustar o funcionamento deste.

Dificuldades como as ilustradas nesta seção, encorajam o desenvolvimento de propostas pedagógicas mais efetivas ao aprendizado de programação. Uma que tem ganho muitos adeptos, por apresentar resultados encorajadores, envolve a utilização do paradigma de programação funcional [JOOSTEN et al 1993][GLASER et al 2000].

3. Abordagens de Ensino de Programação para Iniciantes

Um grande número de teorias de aprendizagem hoje utilizadas são categóricas ao afirmar que a aprendizagem só ocorre com a experiência, é o aprender fazendo. Intuitivamente, os pais auxiliam na educação de seus filhos dessa maneira. O mesmo acontece com o aprendizado de programação, já que esta é uma atividade humana relacionada à resolução de problemas [NÁVRAT 1996], aprendida, através do estímulo à abstração.

Programação é uma tarefa desafiadora e, para amenizar as dificuldades no seu aprendizado, têm sido adotadas diversas abordagens metodológicas para seu ensino em cursos de graduação. Existem muitos relatos de experiência com cursos de programação para quem não tem experiência anterior, mas ainda não se tem dados precisos sobre qual a melhor metodologia a ser empregada e qual o sucesso real desses cursos para os estudantes.

Ao mesmo tempo que aumenta a demanda por cursos de graduação na área, aumenta a oferta de livros e outros textos do tipo "aprenda programação em 3 dias... ou 24 horas". Norvig (2002) comenta que qualquer busca realizada, com esses termos, por esses livros em *sites* de compra produzirá uma quantidade enorme de respostas, o que, segundo ele, só prejudica o entendimento de programação, já que não é possível ser um especialista apenas lendo um livro sobre o assunto. Pesquisas [NORVIG 2002] já mostraram que se leva, aproximadamente, 10 anos para ser um especialista em qualquer área do conhecimento, o que inclui programação.

Outro argumento contra a simples adoção de um livro texto é que os textos apresentam os tópicos em uma ordem puramente conceitual, não fornecendo a experiência necessária ao aprendizado dos princípios e abstrações, fundamentais para a construção de programas. A maioria dos cursos introdutórios atêm-se à apresentação e desenvolvimento de alguns princípios e negligenciam os pontos críticos no aprendizado: resolução de problemas e depuração [McKEOWN e FARRELL 1999].

Como se pode observar, no relatório de referências de currículos de computação para Ciência da Computação [ENGEL e ROBERTS 2001], existem, como referência, estratégias de implementação de cursos introdutórios em Ciência da Computação, utilizando as abordagens: (i) imperativa (prática, com utilização de linguagens como C e Pascal); (ii) orientada a objetos (prática, com utilização de linguagens como C++ e Java); (iii) funcional (prática, utilizando linguagens como Lisp e Haskell); (iv) *breadth* (estudo da computação em vários aspectos teóricos e práticos) ; (v) algoritmos (somente com o estudo de algoritmos em Portugol); e (vi) hardware (visão de hardware utilizados em computação, com utilização de linguagem Assembly). Seguindo qualquer

uma das abordagens mencionadas, implica na complementação do currículo com disciplinas específicas.

4. O Uso da Programação Funcional em Cursos Introdutórios

Dos dois paradigmas mais freqüentemente adotados em cursos introdutórios (imperativo e declarativo), optamos pelo declarativo. A diferença entre os dois é na ênfase. Enquanto no paradigma imperativo a ênfase é em "como" fazer, no declarativo é em "o que" fazer. O paradigma declarativo tem esse nome porque o programador *declara* o que vai fazer, em vez de especificar um método de resolução. As linguagens funcionais estão neste grupo.

Em cursos onde a primeira abordagem é através da programação funcional, o conceito de algoritmo é introduzido com o mínimo de elementos de dispersão [JOOSTEN et al 1993], sem atenção às particularidades de sintaxe das linguagens, como ocorre com as linguagens imperativas (C, Pascal, etc.). O estudante, mesmo no início do curso, está apto a se posicionar acima do mecanismo passo a passo da máquina e declarar a solução de um problema, desde que saiba o método para sua resolução. O sistema operacional, através de um interpretador de linguagem funcional, está apto a interpretar essa declaração para construir a resposta [GLASER, HARTEL et al 2000].

Se, como vimos na seção anterior, o foco principal de um curso de programação é na resolução de problemas, nada mais natural que a utilização de uma linguagem de programação que permita denotar abstrações adequadamente, fornecendo as ferramentas apropriadas para a construção de programas claros e concisos, que podem ser escritos para expressar o algoritmo, sem ambigüidade [JOOSTEN et al 1993]. Com a utilização de uma linguagem funcional, os estudantes aprendem a manipular abstração como uma ferramenta de projeto e estão aptos a descrever seus problemas formalmente, resolvendo mais problemas e cada vez mais complexos. O sucesso na aprendizagem depende, então, mais das habilidades de abstração e menos das habilidades de codificação.

O grande problema das linguagens imperativas é que elas requerem que o programador especifique um processo ou uma receita para chegar à solução de um problema, enquanto linguagens declarativas não permitem a especificação do processo, baseando-se em uma declaração mais abstrata da solução [GLASER, HARTEL et al 2000]. As linguagens funcionais estão próximas à matemática, sendo, portanto, um veículo ideal para o ensino dos princípios de programação em um curso introdutório. Alguém pode argumentar que a matemática também é tão difícil de aprender quanto escrever programas procedurais. A isto podemos contra-argumentar lembrando que para estudantes de computação isso não se aplica, pois a matemática já é um formalismo incorporado à linguagem e o conceito de função é fundamental para a modelagem de problemas do mundo real. Assim, ao invés de o aluno ter dois novos desafios, conhecer um modelo computacional e utilizar o computador, retardamos um deles, visto que os primeiros passos serão dados usando um formalismo familiar.

Os currículos de computação que adotam a linguagem funcional para seus cursos introdutórios o fazem por diferentes motivos, dentre eles, Glaser e Henderson (1995) destacam três atributos compartilhados por todas as linguagens funcionais: (i) são de nível mais alto que as linguagens de programação tradicionais; (ii) são mais próximas da especificação; e (iii) não têm efeitos colaterais na avaliação, ou seja,

variáveis declaradas não mudam de valor durante a execução. Além desses pontos, Hughes (1990) acrescenta que o uso de linguagens funcionais dispensa o programador de prescrever explicitamente o fluxo de controle.

Uma outra linha de pensamento é a defendida por [THOMPSON e HILL 1995], que considera a programação imperativa complementar à declarativa, sendo que esta última fornece uma perspectiva valorosa na outra em muitas formas, dentre as quais: (i) uma linguagem funcional é uma linguagem útil de projeto para programas imperativos, especialmente aqueles que manipulam estruturas de dados dinâmicas; (ii) a abordagem diferente da programação funcional pode tornar comum o que está acontecendo na linguagem imperativa, como as noções de "variável"; e (iii) a abordagem funcional pode também iluminar deficiências nas linguagens imperativas, ou abordagens alternativas que não são familiares a um programador mais tradicional.

Devido ao uso reduzido da programação funcional, ao adotá-la como primeira linguagem não se pode deixa-la isolada no currículo. É necessário que se continue utilizando-a em outras disciplinas, ao longo de todo o curso, pois se corre o risco de os estudantes enxergarem sua utilização apenas como mais uma linguagem que devem aprender, e não compreenderem sua utilidade para a resolução de alguns tipos de problemas. Em [SABRY 1999] sugere-se que se enriqueça o currículo com um pequeno número de cursos estratégicos.

A chave para a programação declarativa é encontrar e desenvolver as abstrações corretas [SABRY 1999]. Portanto, em [GIEGERICH, HINZE et al 1999] recomenda-se a escolha de assuntos de caráter intrinsecamente formal. Eles precisam ser fáceis de serem representados por esqueletos, mas ricos o bastante para serem interessantes.

Nesse contexto, nos cursos descritos em [GIEGERICH, HINZE et al 1999], os temas escolhidos foram Música e Genética Molecular. As partituras de música eram escritas em notação *Haskell* (introduzida de maneira rudimentar), no sistema *Haskore*. Esses temas não são exemplos-padrão de cursos introdutórios, possuindo uma natureza semi-formal. Como conseqüência disso, servem de incentivo aos estudantes.

5. Um Estudo de Caso

Nosso curso de referência acontece em uma universidade federal brasileira no primeiro período dos cursos de Ciência da Computação e de Engenharia da Computação. Os matriculados nessas disciplinas introdutórias ingressam em seus cursos todos os anos via processo seletivo. Portanto, estão familiarizados com a notação matemática e problemas geométricos.

Neste ponto vale comentar sobre o perfil dos estudantes inscritos no curso. Em geral, são estudantes que acabaram de concluir o ensino médio. Mas, apesar disso, o grupo é bastante heterogêneo. Em média, cerca de 5% concluiu um curso técnico em informática, 10% já teve algum contato com programação e a grande maioria (cerca de 98%) já utilizou o computador de alguma forma.

Um curso introdutório para esse perfil de estudante não poderia ser muito teórico e nem utilizar uma linguagem de programação que muitos já saberiam. Isto poderia acentuar o desnível entre os estudantes, privilegiando os que já possuíam alguma experiência. Portanto, a opção por linguagem funcional parecia bastante viável, uma vez

que dificilmente se aborda este paradigma em cursos técnicos. É importante notar que a aprendizagem de programação em cursos de nível médio é, em geral, voltada para algoritmos simples e se concentram principalmente em comandos de uma linguagem. Em geral, esses alunos passam por um bloqueio, visto que não dominam adequadamente o conceito de algoritmo, mas acham que sim e, portanto, não se sentem motivados pelo curso de introdução.

O objetivo do curso é, então, envolver o estudante com a aprendizagem de conceitos e métodos básicos para a construção de programas de computador. A abordagem utilizada enfatiza a importância da solução de problemas, através de problemas geométricos, representados em uma linguagem funcional, atualmente, Haskell.

O curso começa com uma pequena introdução de conceitos básicos em computação, apenas para situar o estudante nos temas: computador, programação, linguagens de programação, propriedades de um programa, paradigmas de linguagens de programação e programação funcional. Com essa introdução, logo no início do curso, o estudante já tem contato com temas importantes e uma breve justificativa para o uso de uma linguagem funcional no início de seu curso.

5.1. Por que linguagem funcional?

Os argumentos embasando a utilização de linguagem funcional no primeiro período são enumeradas no ambiente de suporte à aprendizagem utilizado pelos estudantes [GAVA e MENEZES 2001] [GRUPO GAIA 2002] explicitam a não adequação dos outros paradigmas conhecidos:

- O estudante de graduação em Computação tem de 4 a 5 anos para aprender todos os detalhes da área, portanto não se justifica que tudo tenha que ser absorvido no primeiro semestre. O curso de Programação I (ou Processamento de Dados I) é apenas o primeiro passo e não visa formar completamente um programador. Este é o momento de apresentar-lhe os fundamentos e além disso permitir que ele vislumbre a variedade de problemas;
- O paradigma imperativo requer que o aluno tenha um bom entendimento dos princípios de funcionamento de um computador real, pois elas se baseiam, como as máquinas reais, no conceito de mudança de estados (máquina de Von Neumann);
- O paradigma lógico, outro forte candidato, requer o conhecimento de lógica matemática que o estudante ainda não domina adequadamente;.
- O paradigma funcional é baseado num conhecimento que o aluno já está familiarizado desde o ensino médio (funções, mapeamento entre domínios) o qual é ainda explorado em outras disciplinas do ciclo básico;
- Esta vantagem nos permite concentrar nossa atenção na elaboração de soluções e na descrição formal destas;
- O elevado poder de expressão das linguagens funcionais permite que a abrangência do uso do computador seja percebida mais rapidamente. Em outras palavras, podemos resolver problemas mais complexos já no primeiro curso;

- O poder computacional do paradigma funcional é idêntico ao dos outros paradigmas. Apesar disso, ele ainda não é usado nas empresas por vários aspectos. Destes aspectos podemos citar: (i) no passado, programas escritos em linguagem funcionais eram executados muito lentamente; (ii) a cultura de linguagens imperativas possui muitos adeptos no mundo inteiro, o que inevitavelmente cria uma barreira à introdução de um novo paradigma; (iii) há uma crença equivocada de que linguagens funcionais são difíceis de aprender e só servem para construir programas de inteligência artificial;
- A ineficiência das linguagens funcionais em comparação às imperativas tem se reduzido através de alguns mecanismos tais como: *lazy evaluation*, grafo de redução, combinadores.
- Para fazer um programa que **funciona** (faz alguma coisa, não necessariamente o que desejamos) é mais fácil fazê-lo no paradigma imperativo. Para fazer um programa que funciona **corretamente** para resolver um determinado problema é mais fácil no paradigma funcional, pois esse paradigma descreve "o que fazer" e não "como fazer". É preciso entender o problema para descrevê-lo.
- As linguagens funcionais são geralmente utilizadas para processamento simbólico, ou seja, solução de problemas não numéricos. (Exemplo: integração simbólica X integração numérica). Atualmente constata-se um crescimento acentuado do uso deste tipo de processamento.

A crescente difusão do uso do computador nas mais diversas áreas do conhecimento gera um crescimento na demanda de produção de programas cada vez mais complexos. Os defensores da programação funcional acreditam que o uso deste paradigma seja uma boa resposta a este problema, visto que com linguagens funcionais podemos nos concentrar mais na solução do problema do que nos detalhes de um computador específico, aumentando a produtividade.

5.2. O Currículo

Os tópicos abordados no currículo da disciplina Programação I, ilustrados na Figura 1, são comuns aos currículos de referência da SBC [GT1 e GT2 1999] e da ACM [ENGEL e ROBERTS 2001]. O que é inovador na metodologia adotada neste curso é a utilização de problemas geométricos em cada tópico estudado. Cada tópico é relacionado a um conjunto de exemplos no contexto dos problemas geométricos. Assim, identificamos duas categorias iniciais de problemas: (i) de determinação, aplicados antes de se abordar listas; e (ii) de pertinência, aplicados antes de estruturas de repetição (IF).

```
1. Conceitos Básicos
2. A Linguagem de Programação Haskell
3. A Arte de Resolver Problemas
4. Abstração, Generalização, Instanciação e Modularização
5. Tipos de Dados Numéricos
6. Expressões Lógicas e o Tipo Boolean
7. Definições Condicionais
8. O Teste de Programas
9. Resolvendo Problemas - Os Movimentos do Cavalo
```

```
10.Tuplas
11.Validação de Dados
12.Listas
13.Resolvendo Problemas com Listas
14.O Paradigma Aplicativo
15.Processamento de Cadeias
16.O Paradigma Recursivo.
```

Figura 1. Tópicos Abordados em Programação I

5.3. Problemas Geométricos

A idéia de explorar os problemas geométricos utilizando figuras, onde os alunos são convidados a descrever regiões, nos permite explorar com facilidade os problemas de verificação e os problemas de determinação. Como o estudante já está familiarizado com esse tipo de problema o trabalho principal se concentra na modelagem dentro do paradigma funcional. Além disso, temos observado que o uso de figuras geométricas apropriadas, facilita ao estudante a identificação de abstrações significativas de natureza visual. Na Figura 2, apresentamos exemplos de construções geométricas utilizadas. Construções desta natureza podem ser explorados na determinação de área e verificação de pertinência de um ponto à construção.

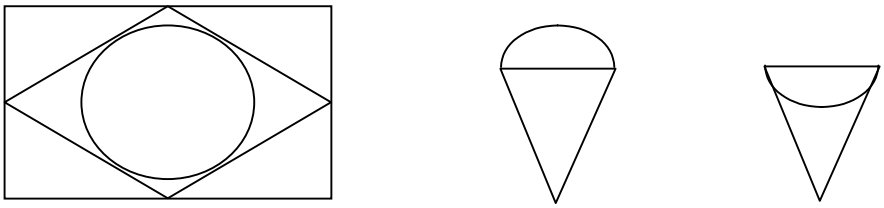

Figura 2. Exemplos de construções geométricas

A seguir, apresenta-se um problema geométrico e a discussão de sua solução, onde enfatizamos a identificação de abstrações.

Questão: Descreva uma função que verifique se o ponto P de coordenadas (x, y) pertence à região hachurada na Figura 3a. O quadrado que circunscreve a figura tem lado L e o quadrado interno, que circunscreve à flor, tem lado L/2. É dado o ponto correspondente ao vértice superior esquerdo do quadrado que circunscreve a figura e o lado L.

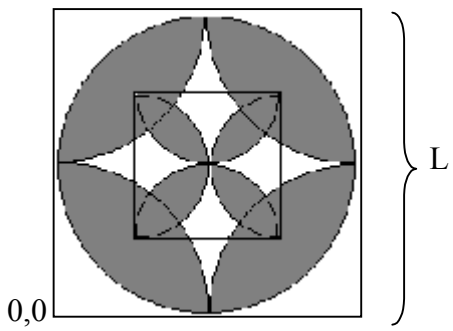

Figura 3a. Um problema de pertinência

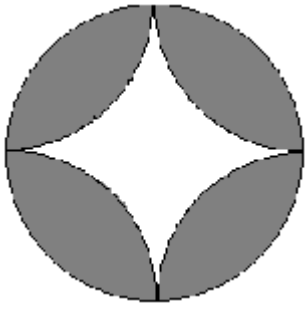

Figura 3b. Sub-figura "flor"

Sugestão de Solução: Importa destacar neste exemplo que o uso de figuras geométricas facilita a busca pela identificação de abstrações, que é, como sabemos um dos pontos cruciais na aprendizagem de programação. Existem muitas possibilidades de construir a solução dessa questão. Uma que nos parece interessante, consiste em identificar figuras que se constituam em abstrações que façam sentido no mundo real. Por exemplo, a figura menor (quadrado) na primeira figura, é uma sub-figura da figura original, que nos parece bastante relevante, vamos chamá-la de **bordas**. A figura que resta (Figura 3b) também é bastante natural, vamos chamá-la de **flor.** Para um ponto pertencer à figura, basta pertencer à **bordas** ou pertencer à **flor.** A descrição de pertinência à bordas pode ser descrita como a pertinência a uma das quatro subfiguras que compõem a borda e para pertencer à flor podemos descrever através da pertinência a uma das pétalas. Aqui é importante observar que todas essas pertinências podem ser descritas usando a pertinência a círculos. A Figura 4 mostra a codificação em Haskell.

```
pFigura  x y vx vy l   =     pBordas x y  a b  l
                          || pFlor x y a b (l/2)
                                    where
                                              a = vx + l/2
                                              b = vy - l/2
pBordas x y a b r      =      pBorda x y a b r  a1  b1 r
                          ||  pBorda x y a b r  a2  b2 r
                          ||  pBorda x y a b r  a3  b3 r
                          ||  pBorda x y a b r  a4  b4 r
pFlor x y a b r        =      PPetala x y a1  b1  r
                          ||  pPetala x y a2 b2  r
                          ||  pPetala x y a3 b3  r
                          ||  pPetala x y a4 b4  r
pCirculo x y  a b r    =          ( x-a )^2 + ( y-b )^2 <= r^2
```

Figura 4. Problema de pertinência em Haskell

5.4. Problemas Avançados

Com uso de problemas geométricos o aproveitamento do curso é superior, possibilitando a resolução de projetos mais complexos como: consulta a banco de dados usando álgebra relacional, técnicas de busca (pesquisa binária), algoritmos de ordenação (*merge*, *sort*, *quick sort*, etc), recuperação de informação baseada em casamento de padrões.

Ao longo do curso, um conjunto de habilidades vai sendo adquirido e utilizado na solução das questões propostas. Dentre tais habilidades, destacamos a facilidade no tratamento de listas e a naturalidade do paradigma recursivo. A Figura 5 apresenta exemplos de construções exploradas no curso.

```
--
-- ordena duas listas pela intercalação da ordenação de
-- suas duas metades
--
mergesort xs = if null (tail xs)
```

```
                then xs
                else intercala (mergesort m) (mergesort n)
                where
                   m = take k xs
                   n = drop k xs
                   k = div (length xs) 2
--
-- intercala duas listas ordenadas
--
intercala xs ys = if (null xs) || (null ys)
                  then xs ++ ys
                  else if head xs <= head ys
                       then head xs : intercala (tail xs) ys
                       else head ys : intercala  xs (tail ys)
```

Figura 5. Definições recursivas para o tratamento de listas

Construções como as ilustradas na Figura 5 tornam-se, por sua vez, componentes para a modelagem e solução de projetos envolvendo um número razoavelmente grande de aspectos do mundo real, como o cenário apresentado a seguir, utilizado em uma das turmas do curso descrito aqui.

Tema: "Banco de Sangue"

Contextualização: Para facilitar o tratamento da demanda por transfusões de sangue, o sistema de saúde criou os chamados "bancos de sangue". Como sabemos, cada transfusão só pode ser realizada usando tipos de sangue apropriados. A adequação de um determinado tipo de sangue é baseada em estudos científicos que identificaram quatro tipos sanguíneos, denominados de 'A', 'B', 'AB' e 'O'. Outros estudos identificaram ainda a existência do chamado "fator RH", que pode ser positivo (+) ou negativo (-), assim o sangue de qualquer indivíduo é classificado de acordo com esses dois atributos. Em um Banco de Sangue, diariamente são feitas doações por pessoas de diferentes tipos sanguíneos, para as quais é feito um registro contendo o número da carteira de identidade do doador (RG), o sexo (S), a data da doação (DD), a data de nascimento (DN), o tipo sanguíneo (TS), o fator RH (RH) e a quantidade doada (QD) (250 ou 500ml). O sangue doado é guardado em recipientes com uma capacidade fixa (250 ml). Também diariamente são feitas requisições pelos hospitais (H), cada requisição indica as características do sangue (tipo e fator RH) e a quantidade solicitada (QS). Sabemos que homens e mulheres possuem intervalos de tempos diferentes para fazer doações. Para homens, o intervalo mínimo é de 2 (dois) meses e para as mulheres é de 3 (três). A idade máxima para doadores é de 60 anos e a idade mínima é 15 anos.

Estruturas: doação: (RG, S, DD, DN, TS, RH, QD)

requisição: (H, TS, RH, QS)

onde RG, QD e QS são do tipo número, DD e DN são triplas na forma (dia, mês, ano), S e RH são do tipo char e H e TS são strings.

O cenário descrito nesse exemplo dá origem a várias questões instigantes e desafiadoras aos estudantes, permitindo que os mesmos elaborem suas soluções

utilizando recursos de programação avançados, refinando suas heurísticas para a construção de soluções de problemas.

6. Considerações Finais

Neste artigo enfocamos as dificuldades encontradas no ensino-aprendizado de programação, abordagens de cursos introdutórios e um estudo de caso, que utiliza linguagem funcional como primeira linguagem, utilizando problemas geométricos. A abordagem permite que haja uma concentração maior na busca por solução de problemas ao invés da clássica dificuldade de entender o modelo computacional das máquinas de Von Newman. A experiência e o convívio com os alunos no segundo curso de programação nos permite afirmar que a maturidade é maior que a de alunos que no primeiro curso utilizam uma linguagem procedural. Além disso temos observado (ainda que sem medir) que a quantidade de alunos que atingem o final deste curso com sucesso é maior do que os obtidos em cursos baseados em linguagem procedural.

Os estudantes submetidos aos cursos descritos não apresentam dificuldades quanto à representação de problemas e conseguem atingir níveis de abstração que os torna aptos a resolverem problemas mais complexos em computação. Isso sem mencionar que, com a utilização do método descrito, há um nivelamento na turma.

No momento estamos desenhando experimentos para aplicar às próximas turmas de Programação I e Processamento de Dados I, a fim de que possamos avaliar o índice preciso de aprendizado nos iniciantes.

Referências

Dijkstra, E. (1982). "On the Teaching of Programming, i.e. on the Teaching of Thinking". In: Selected Writings on Computing: A Personal Perspective. Springer-Verlag NY.

Engel, G. e Roberts, E. (eds) (2001). "Final Report on Computing Curricula 2001 – Computer Science". In: ACM Journal of Educational Resources in Computing, Vol. 1, No. 3. Dezembro.

Gava, T. e Menezes, Crediné (2001). "Moonline: Um Ambiente de Aprendizagem Cooperativa Baseado na WEB para Apoio às Atividades Extraclasse". In: Anais do XII Simpósio Brasileiro de Informática na Educação. pp. 217-224. Novembro.

Giegerich, R.; Hinze, R. e Kurtz, S. (1999) "Straight to the Heart of Computer Science via Functional Programming". In: Proceedings of the Workshop on Functional and Declarative Programming in Education. Setembro, Paris.

Glaser, H.; Hartel, P. ; Leuschel, M. e Martin, A. (2000) "Declarative Languages in Education". Technical Report DSSE-TR-2000-1. Declarative Systems and Software Engineering Group. University of Southampton. UK. Janeiro.

Glaser, H. e Henderson, P. (1995) "Functional Programming and Software Engineering". In: http://www.ecs.soton.ac.uk/~hg/butterworth.ps.gz. To appear in: Software Engineer's Reference Book. Visitada em Março, 2002.

Grupo de Trabalho 1; Grupo de Trabalho 2 (1999). "Currículo de Referência para Ciência da Computação". In: http://sbc.org.br/educacao/cr99.pdf. SBC.

Grupo Gaia (2002). “Programação para Principiantes: Uma Abordagem Funcional”. In: http://www.moonline.ufes.br/reenge/prog_func/intro_PF.html. Visitada em Março.

Joosten, S.; van-den-Berg, K. e van-der-Hoeven, G. (1993). “Teaching Functional Programming to First-Year Students”. In: Journal of Functional Programming 3(1):49-65. Cambridge University Press. Janeiro.

Mckeown, J. e Farrell, T. (1999). “Why We Need to Develop Succcess in Introductory Programming Courses”. In: CCSC – Central Plains Conference, Maryville, MO.

Sabry, A. (1999) “Declarative Programming Across the Undergraduate Curriculum”. In: Proceedings of the Workshop on Functional and Declarative Programming in Education. Setembro, Paris.

Thompson, S. e Hill, S. (1995) “Functional Programming Through the Curriculum”. In: Functional Programming Languages in Education, number 1022 in Lecture Notes in Computer Science, pages 85-102. Springer-Verlag, Dezembro.