UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Dissertação de Mestrado

A Formação e a Prática do Profissional de Programação Incluindo a Discussão de Estratégias para a Aprendizagem Continuada

por

Dênio Mariz Timóteo de Sousa

Campina Grande – PB, Outubro de 1998

UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Dênio Mariz Timóteo de Sousa

A Formação e a Prática do Profissional de Programação: Discussão de Estratégias para a Aprendizagem Continuada

Dissertação apresentada ao curso de Mestrado em Informática da Universidade Federal da Paraíba, em cumprimento às exigências para obtenção do grau de Mestre

Orientador: **Prof. Francisco Vilar Brasileiro**

Campina Grande – PB, Outubro de 1998

S725t Sousa, Dênio Mariz Timóteo de.
A formação e a prática do profissional de programação : discussão de estratégias para a aprendizagem continuada / Dênio Mariz Timóteo de Sousa. - Campina Grande, 1998.
102 f.

Dissertação (Mestrado em Informática) - Universidade Federal da Paraíba, Centro de Ciências e Tecnologia, 1998.
"Orientação : Prof. Dr. Francisco Vilar Brasileiro".
Referências.

1. Programação Computacional - Formação Continuada. 2. Formação Profissional. 3. Aprendizagem Continuada. 4. Dissertação - Informática. I. Brasileiro, Francisco Vilar. II. Universidade Federal da Paraíba - Campina Grande (PB). III. Título

CDU 004.43:378.2(043)

A FORMAÇÃO E A PRÁTICA DO PROFISSIONAL DE PROGRAMAÇÃO INCLUINDO A DISCUSSÃO DE ESTRATÉGIAS PARA A APRENDIZAGEM CONTINUADA

DÊNIO MARIZ TIMÓTEO DE SOUSA

DISSERTAÇÃO APROVADA EM 23.10.98

PROF. FRANCISCO VILAR BRASILEIRO, Ph.D
Presidente

PROF. MARCUS COSTA SAMPAIO, Dr.
Examinador

PROFª ANA CAROLINA BRANDÃO SALGADO, Drª
Examinadora

CAMPINA GRANDE - PB

Agradecimentos

Ao professor *Severino do Ramo de Paiva*, pela colaboração na elaboração das pesquisas contidas neste trabalho.

Aos psicólogos *Joaquim Olimpio Silveira Carvalho Filho* e *Luiz Célio Rangel*, pelos esclarecimentos e sugestões na elaboração dos formulários das pesquisas desenvolvidas neste trabalho.

Ao professor *Camilo de Lélis Medeiros*, DSC, UFPB, pelos comentários e sugestões quanto à direção deste trabalho.

Ao professor *Marcus Costa Sampaio*, DSC, UFPB, pelos comentários e sugestões tecidos sobre a análise dos resultados das pesquisas desenvolvidas neste trabalho.

Ao professor *Paul Brna*, Computer Based Learning Unit, Universidade de Leeds, Inglaterra, pela atenção em nos enviar uma seleção de relevantes artigos de sua autoria sobre os mecanismos cognitivos aprendizado de programação.

Ao professor *Derek Andrews*, University College Northampton, Inglaterra, pelos comentários tecidos sobre "Ensino de Programação vs Ensino de Linguagens".

Ao professor *Gustavo Motta*, do Departamento de Informática, Campus I, UFPB, pela grande ajuda na discussão sobre os paradigmas de programação.

Aos professores e instituições participantes da pesquisa sobre "O Ensino de Programação nas Instituições de Ensino Superior Brasileiras" pelos seus comentários e aos profissionais que participaram da pesquisa sobre "A Prática do Profissional de Programação", sem a ajuda dos quais este trabalho não teria sido possível.

Ao meu orientador, professor *Francisco Brasileiro*, do Departamento de Sistemas e Computação, Campus II, UFPB, pela dedicação do seu tempo, pela colaboração efetiva neste trabalho e pela sua crença na possibilidade de se construir ambientes de ensino cada vez melhores.

À minha esposa *Isabel* e ao meu filho *Pedrinho* pela paciência e imensuráveis colaboração e estímulo.

Aos meus pais, que me fizeram ver o valor da educação e do trabalho.

Resumo

Este trabalho trata de três aspectos fundamentais na vida do profissional de computação, focando particularmente a área de programação: i) sua formação; ii) sua prática no mercado de trabalho; e iii) os mecanismos usados para sua atualização técnica. São apresentados os resultados de duas pesquisas realizadas no âmbito da *World Wide Web*. A primeira pesquisa, denominada "O ensino de programação nas instituições de ensino superior brasileiras", teve como público alvo os profissionais docentes da área de informática e o objetivo de conhecer os aspectos relacionados com o ensino de programação, as linguagens e paradigmas de programação mais comumente ensinadas, opiniões sobre a eficácia didática dos conteúdos teóricos e práticos, além de relatos de experiências de professores e coordenadores de cursos.

A segunda pesquisa abordou o tema "A prática dos profissionais de programação" e foi dirigida aos egressos de cursos de graduação na área de informática ou afins, atuais profissionais, com a intenção de conhecer a realidade do profissional após sua formação, suas dificuldades e os mecanismos usados para sua atualização.

Os resultados das duas pesquisas são discutidos e correlacionados, enquanto buscamos identificar os mecanismos cognitivos envolvidos no aprendizado e na prática de programação para a proposição de algumas estratégias que contribuam para a formação mais duradoura do profissional, e para que o mesmo adote uma conduta eficaz frente à dinamicidade da área e incorpore o "aprendizado" como um elemento importante em sua vida profissional.

Abstract

This work approaches three basic aspects of the life of a computing professional, focusing particularly on the area of programming: i) its education; ii) its practice in the market place; and iii) the mechanisms used for updating its technical knowledge. The results of two surveys carried out in the World Wide Web are presented. The first survey named "Teaching computer programming in the Brazilian Universities", was answered by lecturers of the area with the objective of knowing the aspects related with the programming teaching, the languages and programming paradigms more commonly taught, opinions about the didactic effectiveness of the theoretical and practical contents and reports of teachers' experiences.

The second survey, named "Programming practices of programmers", was directed to the current professionals graduated from courses in computer science and other related areas with the intention of knowing the professional's reality after its formation, its difficulties and the mechanisms used for its modernization.

The results of the two surveys are discussed and correlated, while we looked for to identify the cognitive mechanisms involved in the learning and in the programming practice for the proposition of some strategies aiming at contributing to the better education of the professional, to the adoption of an efficient behaviour of the graduate in face of the dynamics of the area, and to the incorporation of "the learning process" as an important element in its professional life.

Sumário

Índice de Figuras

Índice de Tabelas

1. Introdução

O modelo educacional que vivenciamos atualmente enfrenta um desafio que merece especial atenção. A forma como a tecnologia está continuamente mudando o mundo, o volume de novas informações e a velocidade com que novos conceitos tecnológicos surgem em algumas áreas de conhecimento estão pondo em cheque a agilidade com que as escolas em geral e as Instituições de Ensino Superior – IES – em particular atualizam seus currículos e se adaptam às mudanças. Da mesma forma, o ensino de computação, mais especificamente, também enfrenta este problema [Astrachan-96a]. A adequação dos conteúdos curriculares é um elemento primordial na formação de um bom profissional e na sua inserção no mundo do trabalho, venha a ser ele um pesquisador, um profissional orientado à indústria ou mesmo um profissional que atuará em outra área, desenvolvendo atividades multidisciplinares.

A prática da ciência da computação está mudando rapidamente [Proulx-91]. Novos desenvolvimentos em hardware e software estão mudando a forma como projetamos sistemas e criamos aplicações. Como exemplo, uma aplicação típica mudou de interface baseada em caracteres usando único processador para aplicações com GUI – *Graphical User Interface* – sofisticada, orientadas a eventos e com intrínsecas interações com outras aplicações, inclusive de forma distribuída. Isso ocorreu no intervalo de apenas alguns anos, no qual grande parte dos cursos de graduação teve dificuldade para suportar os novos conteúdos curriculares exigidos, quando conseguiram fazê-lo. Se considerarmos ainda o intervalo de tempo decorrido entre o ingresso de um estudante em um curso de graduação e sua chegada ao mundo do trabalho, quanto do conteúdo curricular haverá mudado ou terá se tornado defasado?

A redução do tempo entre atualizações curriculares dos cursos é, sem dúvida, um fator importante para afinar os seus conteúdos com a realidade do mundo do trabalho, mas não pode ser tomada como uma solução estanque. A má aplicação deste mecanismo e a condução de reformas curriculares constantes podem resultar em grades curriculares confusas, adaptações perigosas para enquadramento de alunos com curso já em andamento, gerenciamento de grades curriculares paralelas e um alto custo de planejamento e execução das mudanças.

O dinamismo do currículo é importante e aumenta as garantias de inserção do profissional no mundo do trabalho. Entretanto, tão importante quanto inseri-lo no mundo do trabalho com conhecimentos compatíveis com as funções que desempenhará, é prepará-lo para enfrentar as mudanças que irão ocorrer a sua volta após a sua desvinculação dos mecanismos formais de ensino.

Preocupados com o excesso de direcionamento dos currículos de computação para o mercado, especialistas em educação ressaltam que em muitas instituições a demanda pelo mercado exerce um papel importante no projeto dos currículos, sendo uma prática comum entre muitos empregadores o envio para as universidades locais de uma lista de itens que eles gostariam que novos graduados tivessem conhecimento (ou potencial para conhecer) para serem contratados [Bruce-96].

Todo curso prepara seu aluno em direção a um perfil predefinido e para atuar em um determinado mercado de trabalho, seja este a indústria, a academia ou a pesquisa. Sendo assim, cursos cujas grades curriculares não estejam em sintonia com aquele mercado para o qual pretendem formar profissionais tendem, naturalmente, a desaparecer. Entretanto, antes de adotar currículos excessivamente voltados para o mercado de trabalho, há a necessidade de preocupar-se também com a empregabilidade desse profissional, ou seja, com o perfil e os conhecimentos básicos que irão sustentá-lo diante das suas necessidades de adaptação às mudanças a longo prazo. É preciso garantir que nossos estudantes possam se manter atualizados e produtivos a despeito das inevitáveis mudanças que ocorrerão na sua área ao longo de suas carreiras.

A preocupação com a definição de conteúdos curriculares não é recente. Em 1968, a ACM – *Association for Computing Machinery*, instituiu um comitê denominado "*Education Board*" e, através dele, publicou pela primeira vez um relatório chamado "Curriculum '68", com recomendações sobre programas e disciplinas que poderiam ser abordados em cursos de graduação em ciências da computação [ACM-68]. Esse relatório foi atualizado em 1979, uma vez que foram reconhecidos os consideráveis avanços ocorridos tanto no campo da computação quanto no da pedagogia [Austing-79]. Durante esse período, inúmeras discussões foram travadas entre os membros do "Education Board", da ACM e os membros do *IEEE Computer Society*, a respeito de uma melhor definição da ciência da computação e da necessidade de unir os esforços das várias entidades preocupadas com o assunto. Em conseqüência, foi criado em 1985 um comitê chamado *The Joint ACM/IEEE-CS Curriculum Task Force*, composto por cientistas em computação e renomados educadores com o objetivo de estabelecer uma definição da "computação como disciplina" e apresentar recomendações quanto à seqüência das disciplinas introdutórias que iriam formar um embasamento para os currículos na área, tendo publicado um relatório em 1988. A força tarefa continuou trabalhando na ampliação das recomendações para todo o currículo de computação e produziu, em 1991, o relatório "Computing Curricula 1991", contendo não só uma atualização das recomendações anteriores como também a unificação de um conjunto de recomendações oriundas de duas das maiores sociedades que representam a disciplina

de computação, nos seus vários contextos acadêmicos ligados à ciência e à engenharia: o IEEE e a ACM [Tucker-91].

No Brasil, a Sociedade Brasileira de Computação – SBC – é o principal órgão que desenvolve esforços para acompanhar as evoluções no campo do ensino da computação, através da sua Comissão de Assuntos de Ensino, que tem publicado recomendações para o ensino de computação, como o Currículo de Referência para Cursos de Graduação Plena em Computação [Nunes-94], publicado em 1991 e o CR96 – Currículo de Referência para Cursos de Graduação em Computação [Bigonha-96], publicado em 1996. Todos estabelecem matérias essenciais e complementares visando "definir cursos com uma boa e sólida formação básica".

Tanto a ACM e o IEEE-CS quanto a SBC, portanto, reconhecem a necessidade de uma formação fundamental ampla para garantir a sobrevivência profissional em uma área de transformações aceleradas [Tucker-91], [Bigonha-96].

Dentro do currículo de computação, em seus diversos contextos (processamento de dados, ciência da computação, engenharia da computação e outros), observamos a importância do papel da matéria *programação*, devido a sua permeabilidade nas diversas outras matérias do currículo. Conceitos de programação são usados como parte dos processos de projeto de software, na implementação de modelos e muitas vezes nos processos de prova de resultados teóricos, abrangendo todos os componentes fundamentais do currículo [Tucker-91].

Segundo alguns especialistas, a metodologia de ensino que introduz os conceitos de programação aos milhares de estudantes a cada ano é dominada por uma coleção de linguagens de programação freqüentemente ultrapassadas, onde predomina a solução de pequenos problemas, ausência de trabalho em equipe e ignorância de métodos formais [Tucker-96a]. Metodologias atuais, como projeto orientado a objeto e interfaces orientadas a eventos, solução de partes modulares de grandes problemas, o papel das equipes de trabalho, uso de modernas ferramentas de teste e certificação de qualidade em software, ainda estão longe do alcance de boa parte dos currículos que pretendem ensinar nossos estudantes a entender os princípios modernos da ciência da computação e da engenharia da computação.

Devido ao papel multidimensional que a programação exerce no currículo de computação, principalmente em sua fase inicial, e na necessidade de estabelecer mecanismos que contribuam para a formação duradoura do profissional, acreditamos na necessidade de direcionar esforços para investigar algumas alternativas metodológicas para o ensino de

programação que possam contribuir para a formação do estudante e, consequentemente, do profissional de programação.

Este trabalho está dividido em duas partes. Na primeira parte, composta pelos capítulos 2 e 3, descrevemos e comentamos duas pesquisas realizadas sobre a disciplina programação, com o intuito de conhecer a realidade do ensino de programação e da prática do profissional formado pelos cursos da área de informática e áreas afins. No capítulo 2, apresentamos a primeira pesquisa, denominada "O ensino de programação nas instituições de ensino superior brasileiras", que teve como público alvo os profissionais docentes da área de informática e como objetivo conhecer os aspectos relacionados com o ensino de programação, as linguagens e paradigmas de programação mais comumente ensinadas, opiniões sobre a eficácia didática dos conteúdos teóricos e práticos, além de relatos de experiências de professores e coordenadores de cursos.

No capítulo 3 apresentamos a segunda pesquisa, denominada "A prática dos profissionais de programação", a qual foi dirigida aos egressos de cursos de graduação na área de informática e afins, atuais profissionais. Nossa intenção era conhecer a realidade do profissional após sua formação, mantendo o foco nas atividades ligadas à programação. Ambas as pesquisas fizeram uso dos recursos da Internet para divulgação e coleta de dados (correio eletrônico e *World Wide Web*).

A segunda parte deste trabalho contém os capítulos 4 e 5, intitulados "Discussão" e "Conclusão", respectivamente. No capítulo 4 apresentamos uma discussão acerca do que aprendemos com as pesquisas sobre o uso de linguagens no ensino de programação, sobre modernização curricular e sobre a dificuldade de atualização que os profissionais egressos relataram enfrentar em sua prática profissional diária. O relato sobre o que aprendemos é finalizado com uma avaliação crítica sobre a metodologia, a implementação dos formulários de pesquisa e uma reflexão sobre falhas que poderão ser evitadas em futuros trabalhos nessa direção.

Ainda no capítulo 4, fazemos uma incursão na área de psicologia cognitiva, em busca da identificação dos mecanismos cognitivos envolvidos no aprendizado e na prática de programação. Sempre guiados pelas preocupações identificadas nas pesquisas sobre o ensino e a prática da programação, buscamos ampliar nosso conhecimento sobre como se aprende programação, quais as principais dificuldades enfrentadas pelos alunos, a relação existente entre *programação*, *solução de problemas* e *codificação*, bem como as diferenças entre ensinar programação para novatos (formação, educação) e para profissionais (reciclagem, aperfeiçoamento, treinamento). A discussão nos mostra que a atividade de programação é muito mais ampla e complexa do que a tarefa de codificar em uma

linguagem de programação e que ensinar programação para formar profissionais aptos a desenvolverem a aprendizagem continuada deve, portanto, ir além do ensino de linguagens.

Por se tratar de uma compilação de vários trabalhos sobre a relação entre a programação e psicologia cognitiva, a discussão dos mecanismos cognitivos e dificuldades envolvidas no aprendizado de programação é feita inicialmente de maneira isolada da avaliação das pesquisas apresentadas na Parte I deste trabalho, mas culmina com a elaboração de um conjunto de estratégias para melhorar o ensino de programação e para o desenvolvimento de uma cultura que se mostra hoje como de fundamental importância para a adaptabilidade do profissional em seu campo de trabalho: a aprendizagem continuada. As estratégias, portanto, se baseiam na análise dos resultados das pesquisas realizadas e na discussão estabelecida sobre os aspectos cognitivos envolvidos no aprendizado e na prática da programação.

O capítulo 5 resume os principais aspectos observados nas pesquisas sobre o ensino de programação, a prática do profissional, os mecanismos cognitivos do aprendizado de programação e tenta fazer uma ligação com trabalhos futuros na área. Os anexos fornecem material de apoio ao estudo relatado, cujas referências aparecem ao longo do texto.

Este trabalho não se propõe a encontrar todas as respostas para os problemas do ensino de programação e, muito menos sobre o ensino de computação em geral. Entretanto, tentaremos contribuir lançando um foco bem dirigido e delimitado sobre o assunto, em busca de clarear o terreno pouco ainda explorado, principalmente no Brasil, que é o ensino de programação. Nossa pretensão, portanto, é contribuir para o estabelecimento de um marco zero no acompanhamento do perfil do ensino de programação nas IES brasileiras e para o estabelecimento de alguns parâmetros para a avaliação da atividade do profissional egresso.

Para tanto, buscamos a opinião dos professores que atuam diretamente no ensino de programação e sintetizamos suas posições sobre o que é importante no ensino de programação, suas experiências de sucesso vivenciadas, informações sobre cursos, disciplinas, métodos e resultados obtidos.

A avaliação da prática do profissional acrescenta uma contribuição no momento em que buscamos coletar subsídios para o estabelecimento de um equilíbrio entre a prática pedagógica e a real necessidade do profissional no mercado de trabalho, sem sacrifício para nenhuma das partes.

Outra contribuição que pretendemos trazer é dirigida aos professores de programação, que muito se fartam de material sobre técnicas e linguagens de programação, mas se ressentem

de material didático que discuta apropriadamente o ensino de programação e os aspectos relacionados ao aprendizado dessa disciplina. A revisão da literatura atualizada, a discussão e a compilação dos principais aspectos sobre mecanismos cognitivos do aprendizado, portanto, pode ser um bom material para o professor de programação, tanto pela fundamentação teórica apresentada como também pela apresentação de algumas estratégias para melhorar o ensino de programação.

Finalmente, ao elaborarmos este estudo, estamos buscando ampliar nosso conhecimento sobre a conexão entre três aspectos fundamentais na vida do profissional de programação: sua formação, sua prática e os mecanismos usados para sua atualização técnica. Nosso objetivo não é, absolutamente, esgotar o assunto, mas lançar mão do *feedback* dos nossos docentes e dos atuais profissionais para estabelecer alguns pontos de reflexão e apontar aspectos relevantes na formação do profissional que podem contribuir para a adoção de uma conduta eficaz do egresso frente à dinamicidade da área e para a incorporação do "aprendizado" como uma constante em sua vida profissional.

Parte I

2. Pesquisa 1: O Ensino de Programação nas Instituições de Ensino Superior Brasileiras

2.1. Apresentação

Alguns esforços têm sido feitos para estabelecer uma certa padronização curricular nos cursos da área de computação, tanto no Brasil como no exterior, conforme comentamos anteriormente. No Brasil, a Comissão de Assuntos de Ensino da SBC tem se preocupado em publicar currículos de referência [Nunes-94], [Bigonha-96], os quais norteiam a criação de novos cursos e as reformulações dos cursos já existentes na área de computação. Entretanto, o escopo dos trabalhos é amplo, uma vez que tece recomendações sobre as várias matérias existentes, e propositadamente superficial em cada uma delas, por se tratarem de modelos.

Mesmo que as instituições de ensino se baseiem em padrões curriculares estabelecidos, elas ainda têm autonomia para implementar variações em seus conteúdos programáticos, fluxo de disciplinas, carga horária e ainda adotar metodologias próprias para o ensino dos conteúdos. No caso de várias matérias, tais como "Algoritmos", "Linguagem de Programação", "Estrutura de Dados" ou "Técnicas de Programação", é possível adotar-se uma ou mais linguagens como veículo para apresentação dos conteúdos e adotar variações metodológicas de acordo com a disponibilidade de recursos humanos ou até mesmo de infra-estrutura.

Além disso, a liberdade que as instituições dispõem para escolher como fazer sua grade curricular, estabelecer conteúdos programáticos e preparar suas aulas, associada às diferenças individuais dos alunos, de formação dos docentes e outros fatores diversos, torna muito difícil uma identificação, à distância, dos elementos que contribuem para melhores resultados em instituições diferentes.

Considerando o interesse deste trabalho, que atua sobre um foco específico do campo do ensino de programação, algumas questões são colocadas: Existe diferença em ensinar programação de uma maneira ou de outra? Que elementos estão por trás do alcance de melhores ou piores resultados, sobre o ensino de conteúdos curriculares semelhantes? Qual o perfil do ensino de programação nas instituições brasileiras?

Para a obtenção de algumas respostas, acreditamos que a opinião dos professores que atuam diretamente no ensino de programação pode trazer grandes contribuições, no sentido de revelar o que é importante no ensino de programação, quais as experiências de sucesso que vivenciaram, o que funciona ou o que não funciona no ensino dessa importante matéria

da computação. Assim sendo, elaboramos uma pesquisa denominada "Ensino de Programação nas Instituições de Ensino Superior Brasileiras", visando obter informações sobre cursos, disciplinas, métodos, resultados obtidos e a opinião dos nossos professores sobre o ensino de programação.

A pesquisa não tem finalidade comparativa e limita-se às instituições de ensino superior brasileiras.

2.2. Objetivos

Esta pesquisa tem o objetivo de conhecer o ambiente que envolve o ensino de programação, considerando os diversos cursos que têm a disciplina programação como componente de sua grade curricular. A pesquisa visa, ainda, estabelecer o perfil do ensino de programação e faz sondagens sobre a estrutura das disciplinas ligadas ao ensino de programação. Por fim, a pesquisa busca a opinião dos professores sobre questões relativas à prática do ensino de programação, tais como adoção de linguagens e paradigmas de programação, relacionamento entre conteúdos de cunho prático e teórico, entre outras.

2.3. Materiais e métodos

A pesquisa foi organizada em um formulário com 5 partes principais, que coletam informações sobre:

1) o participante;
2) o curso no qual leciona;
3) as disciplinas de programação, seus conteúdos e linguagens;
4) opinião sobre afirmativas relacionadas ao tema; e
5) comentários livres sobre experiências de sucesso ou sugestões para o alcance de melhores resultados no ensino de programação.

O formulário contendo as questões da pesquisa foi transcrito para o formato HTML e publicado com livre acesso na Rede Internet, em um servidor *Web* do Departamento de Sistemas e Computação da Universidade Federal da Paraíba (DSC/UFPB). O formulário eletrônico era baseado na tecnologia CGI (*Computer Gateway Interface*), a qual permite a interação de programas funcionando no servidor com um browser remoto (cliente), via HTTP - *Hypertext Transfer Protocol*.

A recepção das respostas dos formulários passava por uma processo de consistência *on line*, que identificava erros de preenchimento nos campos obrigatórios. Em caso de algum erro, os nomes dos campos eram indicados ao participante para que fossem corrigidos, sem

a necessidade de redigitação dos demais. Uma vez que todos os campos fossem aceitos, eles eram armazenados para posterior tratamento.

Na primeira parte do formulário, solicitávamos a identificação do participante, indicando seu nome completo (opcional), seu endereço eletrônico (opcional) sua função atual (obrigatório) e sua experiência, em anos, na área de ensino (obrigatório). A identificação do participante da pesquisa foi importante para eventuais contatos para informações complementares, dúvidas ou verificação da identidade do participante na instituição informada.

Na segunda parte, solicitávamos identificar a instituição de ensino onde leciona e o nome do curso (ambos obrigatórios). Esta informação foi importante para a pesquisa, uma vez que o papel que as disciplinas de programação exercem em cada curso pode variar (ex. Ciência da Computação, Engenharia da Computação). Também solicitávamos o número médio de alunos formados por ano no curso, visando ter uma idéia da participação da instituição/curso na formação de profissionais. A primeira e a segunda parte do formulário podem ser vistas na Figura 1.

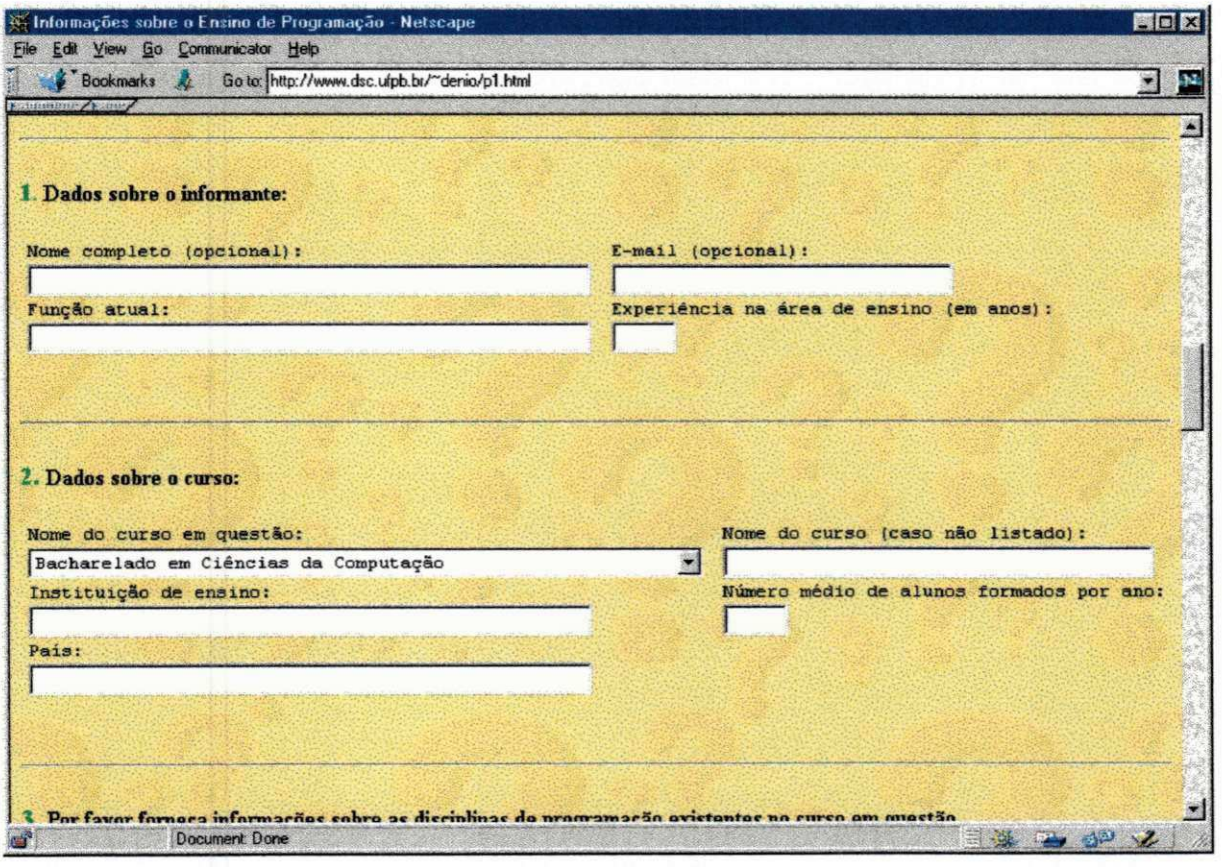

Figura 1 – Primeira e segunda partes do formulário eletrônico da Pesquisa 1: identificação do respondente e do curso

Na terceira parte, solicitávamos informações especificamente sobre as disciplinas de programação no curso em questão. As informações solicitadas eram: nome da disciplina, pré-requisitos, carga horária semanal e uma breve descrição da ementa da disciplina. O

objetivo dessa solicitação era construir uma grade curricular resumida das disciplinas de programação para o curso em questão. A terceira parte do formulário pode ser vista na Figura 2.

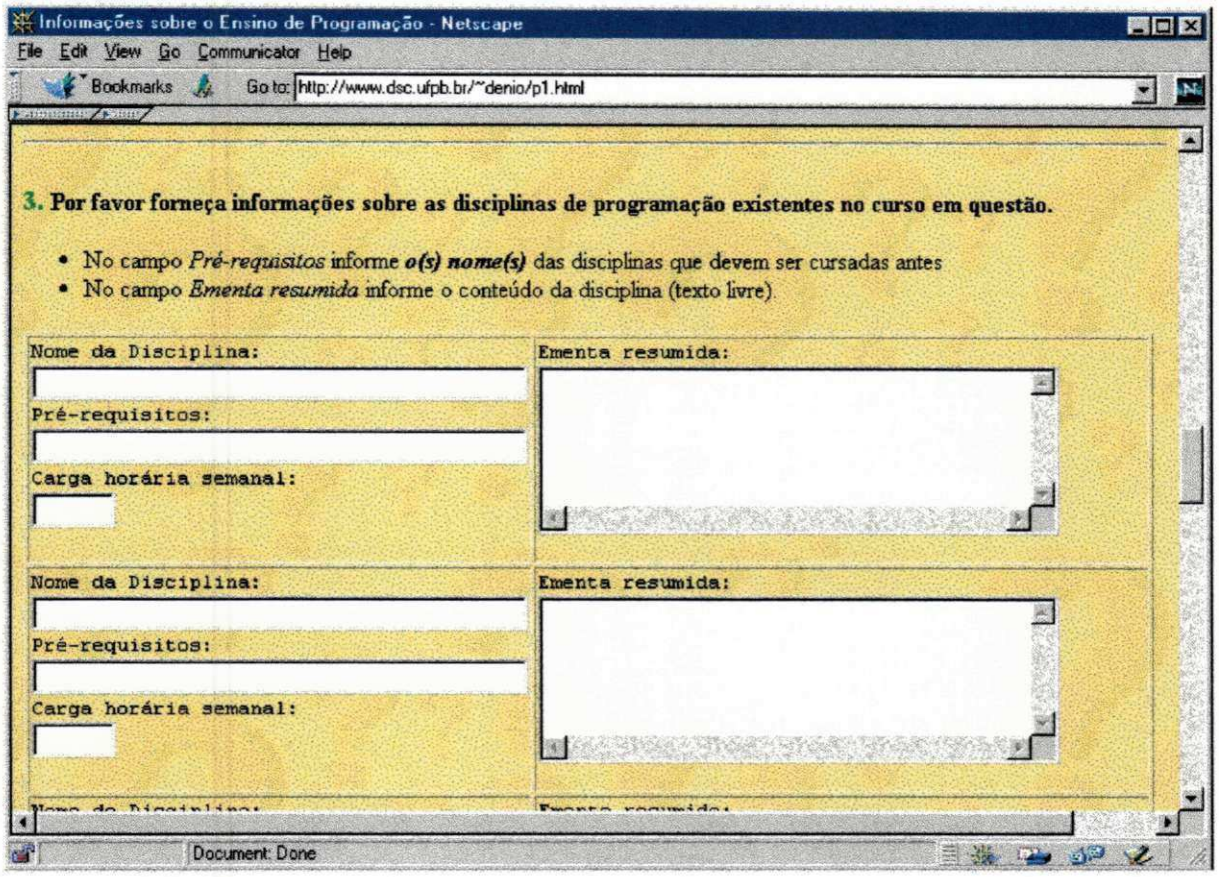

Figura 2 – Terceira parte do formulário eletrônico da Pesquisa 1: Informações sobre grade curricular do curso

Na quarta parte fazíamos algumas assertivas sobre temas ligados ao ensino de programação e solicitávamos ao participante que manifestasse sua opinião de concordância ou discordância, usando a seguinte escala: "*DISCORDO FORTEMENTE*", "*DISCORDO*", "*NÃO SEI*", "*CONCORDO*" e "*CONCORDO FORTEMENTE*". Esta escala foi usada para viabilizar a recepção das opiniões neutras, das opiniões tendenciosas e das opiniões extremas, embora não tenhamos usado ponderações diferentes para os extremos quando da tabulação dos resultados. Nesta parte da pesquisa as respostas eram opcionais, de maneira que o respondente podia não opinar em todas as questões ou até mesmo em nenhuma.

O objetivo de solicitar opiniões dos professores sobre algumas informações era conhecer a visão geral do corpo docente sobre alguns assuntos controvertidos, tais como maior ou menor importância da parte prática e da parte teórica do ensino de programação; a necessidade de padronização de conteúdos nessa matéria; ou a adaptabilidade do profissional (ex-aluno) no mercado de trabalho, após sua formação. A quarta parte do formulário pode ser vista na Figura 3.

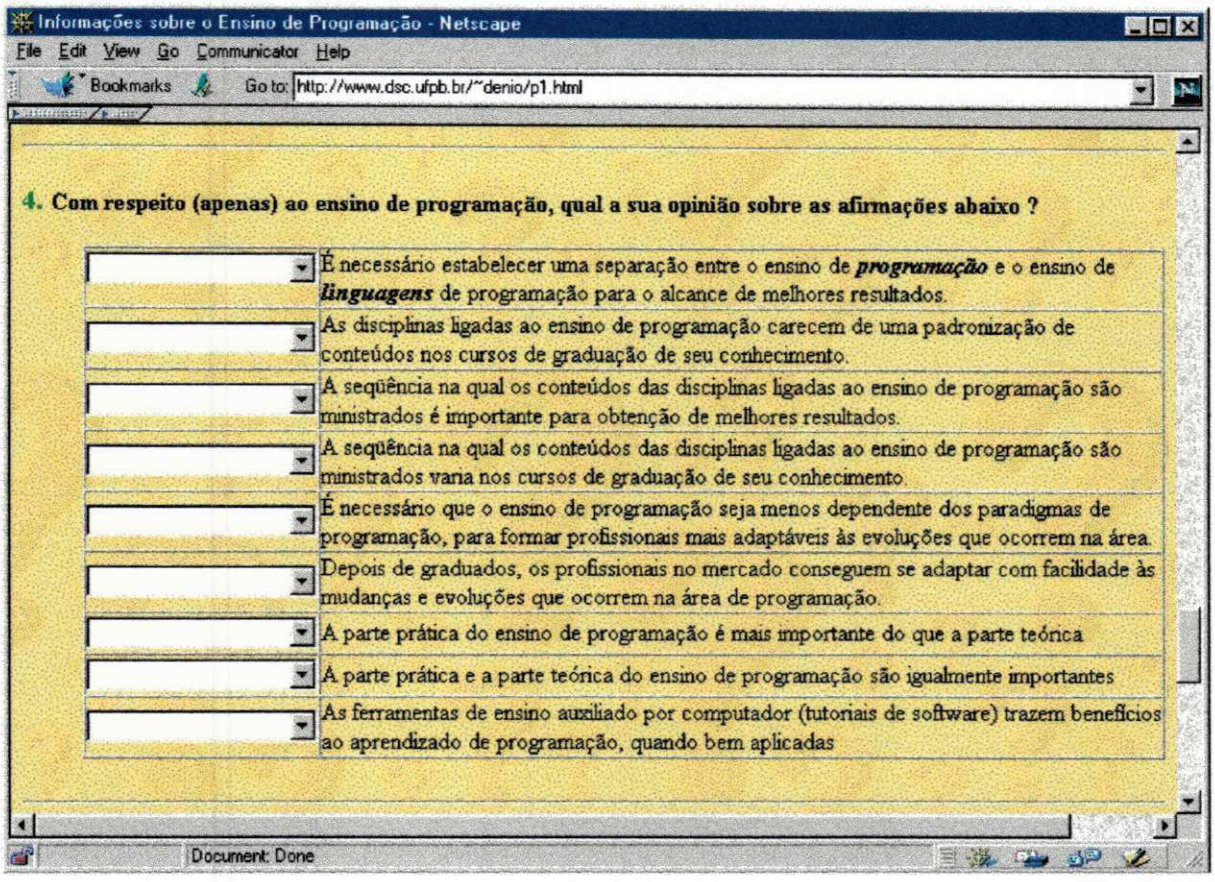

Figura 3 – Quarta parte do formulário eletrônico da Pesquisa 1: opinião do professor/coordenador

Na quinta parte solicitávamos algum comentário ou sugestão sobre a pesquisa ou o relato sobre alguma experiência válida quanto ao alcance de melhores resultados no ensino de programação. Nessa parte, o participante podia escrever um texto livre em um espaço reservado no formulário, sem limite de tamanho. A idéia de coletar essas informações de caráter subjetivo era permitir ao respondente dar uma colaboração extra à pesquisa através de comentários e/ou sugestões. Além disso, quaisquer opiniões que não pudessem se encaixar no formato do formulário ou algum complemento às questões colocadas poderiam ser feitas aqui. A quinta parte pode ser vista na Figura 4.

Para finalizar, solicitávamos ao respondente sua autorização para empregar suas colocações, respostas e comentários no relatório da nossa pesquisa, através de citações ou referência ao participante (veja item 6 do formulário na Figura 4).

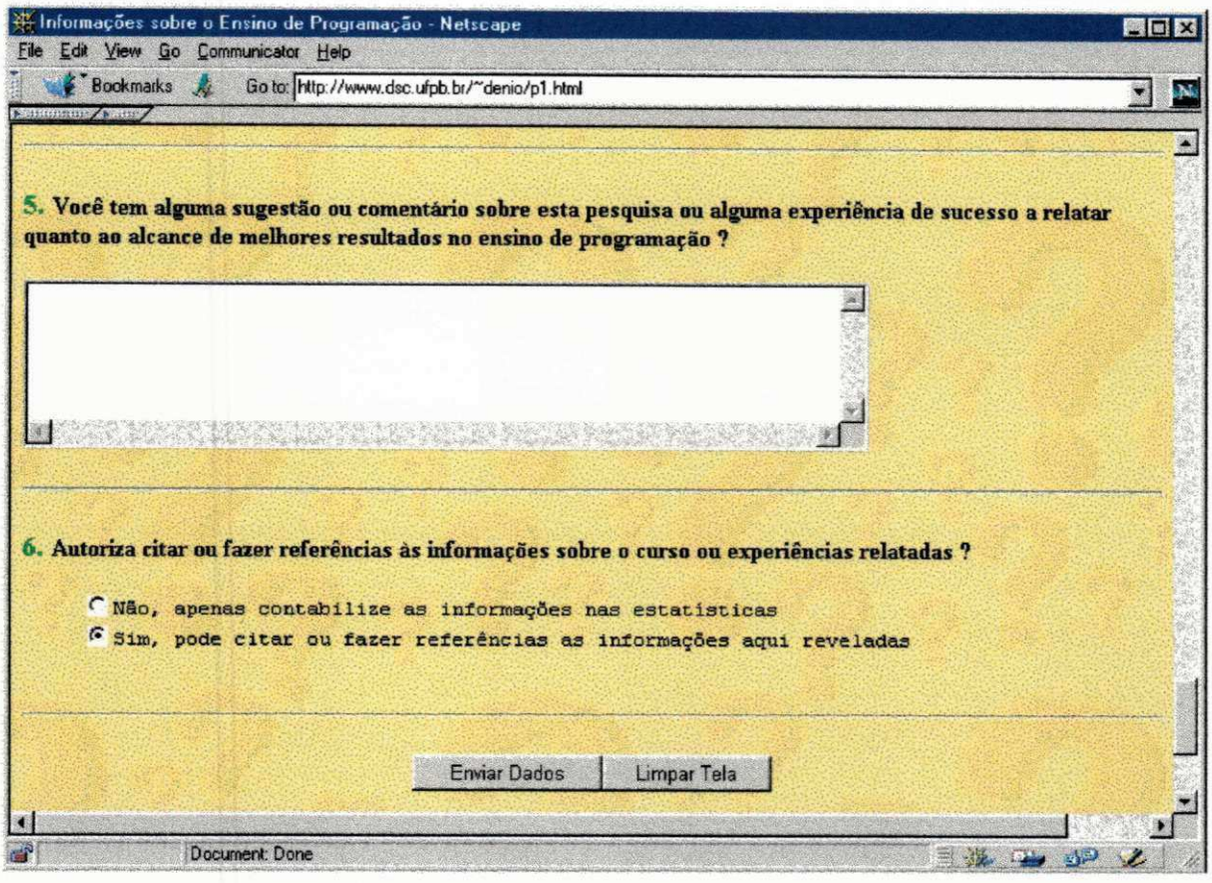

Figura 4 – Quinta e sexta partes do formulário eletrônico da Pesquisa 1: Opiniões, relatos de experiências e autorização para citação nesta pesquisa

Uma cópia completa do formulário pode ser vista no Anexo A.

2.4. O público alvo e a divulgação

Nesta pesquisa, nossa amostragem é baseada no universo de professores ou coordenadores de quaisquer cursos de graduação em cujos currículos existam disciplinas de "programação de computadores". Além disso, o professor deveria atender a pelo menos uma das seguintes condições: 1) estar lecionando atualmente ou já ter lecionado disciplinas de programação; 2) ser ou ter sido coordenador de curso.

A divulgação foi feita no meio acadêmico através de correio eletrônico, com envio de mensagens diretamente aos coordenadores e professores de cursos em todo o País e envio de mensagens para listas de discussão sobre programação, listas de discussão de professores de informática em universidades e listas com temas ligados à educação em informática. Ao todo foram enviadas 1887 mensagens para 94 instituições diferentes (desprezando-se nesta contabilidade as mensagens indiretas produzidas pelas listas de discussão), no período compreendido entre os meses de fevereiro e julho de 1997.

2.5. *Resultados*

Ao todo, a *Pesquisa Sobre o Ensino de Programação* obteve 34 respostas válidas, submetidas a partir de 28 instituições diferentes, todas brasileiras (vide Anexo B). A média do tempo de experiência na área de ensino declarado pelos respondentes foi de 9,3 anos.

Quanto à função que exerciam, todos os respondentes eram professores e em 26% dos casos, eram também coordenadores ou sub-coordenadores de curso e em 7% dos casos eram também pesquisadores.

Dentre os respondentes que completaram a primeira parte do formulário, 30 (86%) informaram o número de alunos formados por ano no curso em questão. Considerando-se os que informaram, o número médio de alunos formados por ano foi de 48,5.

No universo das instituições pesquisadas, a participação dos cursos de graduação concentrou-se basicamente na área de computação, com 97% dos cursos, como pode ser visto na Figura 5. A distribuição dos cursos de graduação pesquisados foi a seguinte: Bacharelado em Ciência da Computação (52%), Processamento de Dados (20%), Engenharia da Computação (19%), Bacharelado em Informática (3%), Informática Industrial (3%), Engenharia Elétrica (3%).

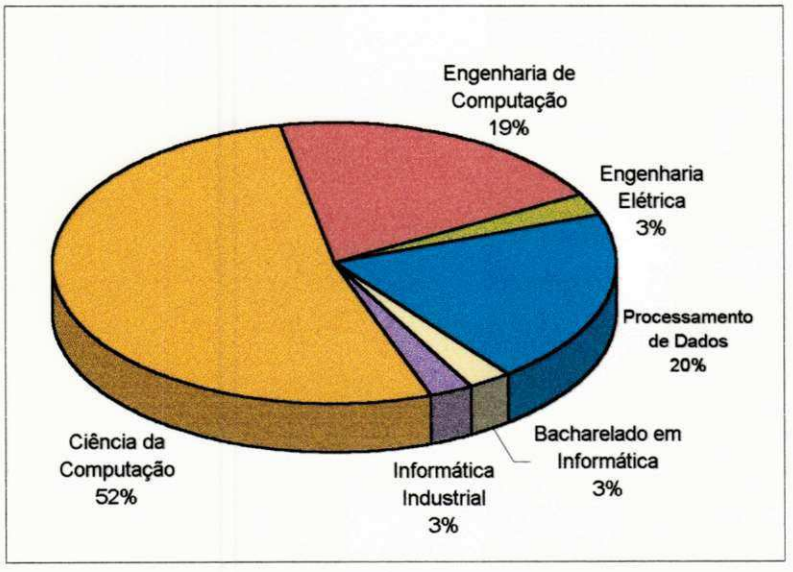

Figura 5 – Distribuição por curso dos participantes da Pesquisa Sobre o Ensino de Programação

Como o instrumento de pesquisa (formulário) era dividido em 5 partes, sendo apenas as duas primeiras partes obrigatórias (dados do informante e da instituição/curso), ocorreu que nem todas as partes foram respondidas em cada formulário retornado. O número de respostas para cada parte será informado individualmente na descrição dos resultados a seguir.

2.5.1. *Sobre o ensino de linguagens e os paradigmas de programação*

Os resultados apresentados nesta seção são baseados nas informações obtidas a partir da terceira parte do formulário da pesquisa.

Inicialmente, esta parte do formulário obteve 18 respostas. Devido ao baixo número de respostas obtidas e ao fato de muitas delas estarem incompletas, a reconstrução de uma grade curricular padrão foi inviabilizada. Sendo assim, praticamente reconduzimos a pesquisa para esta parte, através de acesso a *home pages* das instituições e, quando necessário (e possível), consultas diretas ao respondente por correio eletrônico.

Essa "diligência" nos permitiu obter de forma segura apenas as informações sobre linguagens de programação usadas e a ordem cronológica em que são apresentadas aos alunos, tendo sido ampliado o número de respondentes para 34, considerando-se cursos diferentes. As instituições/cursos participantes são mostradas no Anexo B.

Na tabulação dos resultados, agrupamos as linguagens que mais freqüentemente apareceram nas respostas dos respondentes, permitindo identificar a participação de cada uma delas dentre as linguagens mais ensinadas em primeiro, segundo e terceiro lugares, considerando a ordem cronológica.

Como conseqüência dos dados obtidos sobre as linguagens usadas nas disciplinas, nos foi possível traçar também um perfil para a abordagem dos paradigmas de programação conhecidos como "Imperativo" (ou "Procedural"), "Orientação a Objeto", "Lógico", "Funcional" e "Concorrente" [Watt-91].

LINGUAGEM	PARADIGMA PREDOMINANTE	ENSINADA EM		
		1º LUGAR	2º LUGAR	3º LUGAR
Pascal	Imperativo	**64%**	4%	-
C	Imperativo	15%	**32%**	10%
FORTRAN	Imperativo	6%	7%	-
Clipper	Imperativo	6%	-	-
Smalltalk	Objeto	6%	-	-
COBOL	Imperativo	3%	7%	10%
C++	Objeto	-	21%	**30%**
Prolog	Lógico	-	14%	15%
Pascal OOP / Delphi OOP	Objeto	-	8%	5%
Visual Basic	Objeto	-	4%	-
Haskell	Funcional	-	4%	-
Java	Objeto	-	-	10%
Assembly	Imperativo	-	-	5%
Lisp	Funcional	-	-	5%
ML	Funcional	-	-	5%
Scheme	Funcional	-	-	5%

Tabela 1 – Preferência para o ensino de linguagens de programação em ordem cronológica

Como podemos ver na Tabela 1, as IES brasileiras têm uma grande preferência pelo uso da linguagem Pascal para apresentação dos primeiros conceitos de programação. O fato de Pascal ser considerada uma linguagem com recursos didáticos adequados para lidar com a inexperiência do aluno, por dispor de boa documentação, ser altamente estruturada e fortemente tipada, provendo um ambiente "seguro" para programadores novatos, explica essa liderança. Os argumentos para a adoção da linguagem Pascal, entretanto, são puramente didáticos pois na prática do mercado de trabalho o uso dessa linguagem não tem o mesmo índice de preferência, como concluiremos do item 3.4.2. adiante.

A linguagem C recebe o destaque de segundo lugar, mesmo considerada por grande parte dos docentes como uma linguagem "árida" sob o ponto de vista didático. Sua presença, mesmo que modesta, no conjunto das linguagens ensinadas em primeiro lugar talvez se explique pelas grades dos cursos de "Engenharia da Computação" e "Engenharia Elétrica", nos quais a preocupação com o ensino de linguagens de programação é mais próxima da aplicabilidade prática do que da fundamentação teórica. Das demais linguagens, observamos a presença de Smalltalk, Clipper e FORTRAN empatados em terceiro lugar e COBOL em quarto. Nesta avaliação, podemos constatar que Smalltalk é a única linguagem representante do paradigma "objetos", sendo todas as demais representantes do paradigma "imperativo", totalizando a quase unanimidade de 94%.

Quanto à escolha da segunda linguagem a ser apresentada aos alunos, a Tabela 1 nos mostra uma maior diversidade. A presença das linguagens C, FORTRAN, COBOL e Pascal, representando o paradigma "imperativo" e das linguagens C++, Pascal OOP/Delphi OOP e Visual Basic, representando o paradigma "Objeto", nos mostra uma disputa entre estes paradigmas (50% e 32%, respectivamente), embora o primeiro se mantenha ainda como predominante. Podemos constatar ainda a presença de linguagens representando os paradigmas "lógico" (Prolog) e "funcional" (Haskell). Esta distribuição demonstra uma preocupação com a inserção de classes diferentes de linguagens no currículo, visando diversificar a abordagem dos paradigmas e apresentar visões variadas para solução de problemas.

Para o ensino da terceira linguagem de programação, a escolha recai sobre a linguagem C++, em sua maioria. Juntas, as linguagens C++, Java e Pascal OOP/Delphi OOP levam o paradigma "Objeto" a 45% da preferência das IES brasileiras, enquanto o paradigma "imperativo", composto por C, Assembly e COBOL, alcança 25% do total, ficando em segundo lugar na preferência. O paradigma "lógico" alcança um índice de 15% (Prolog), ficando empatado com paradigma "funcional", o qual é representado pelas linguagens Scheme, ML e Lisp, somando 15%.

A Figura 6 é a representação decorrente do observado na Tabela 1. Para obtenção dos percentuais de cada paradigma, somamos os percentuais das linguagens que os representam.

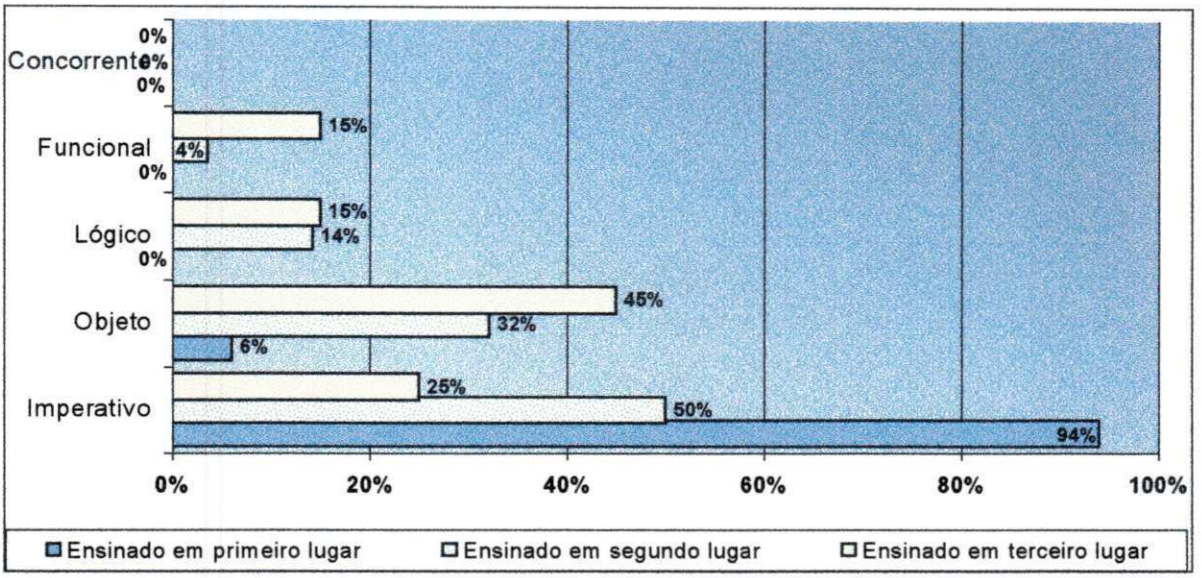

Figura 6 – Demonstrativo dos paradigmas e da ordem em que são ensinados

A apresentação dos conceitos de orientação a objeto só ganha preferência significativa a partir de quando o aluno já dispõe de conhecimento prévio sobre uma linguagem (no caso, da família das linguagens imperativas, com freqüência Pascal). Enquanto as linguagens do paradigma imperativo são apresentadas no início do curso e decrescem na preferência à medida em que o curso avança, as linguagens do paradigma objeto crescem na preferência dos professores enquanto o curso avança.

Os paradigmas "lógico" e "funcional" parecem gozar de uma unanimidade absoluta quanto ao momento em que devem ser apresentados aos alunos: depois de ter sido apresentado o paradigma imperativo ou, na maioria dos casos, depois de trabalhados pelo menos os paradigmas "imperativo" e também o paradigma "objeto".

Foi observado que nenhuma instituição participante da pesquisa atualmente adota oficialmente uma linguagem que trabalhe essencialmente o paradigma "concorrente", a exemplo das linguagens *Ada, Linda* ou *Occam*, entre outras. Talvez o principal obstáculo para o uso mais amplo de concorrência seja sua baixa popularidade e reduzida aplicabilidade em sistemas de ambientes comerciais, alvo de boa parte dos currículos. No entanto, podemos perceber atualmente um crescente uso de computadores de arquitetura paralela com múltiplos processadores, com aplicabilidade até mesmo em ambientes de pequeno e médio portes devido à redução do custo desse tipo de *hardware*. Esses equipamentos têm forçado uma crescente demanda pela construção de softwares mais

inteligentes e robustos, com capacidade de explorar com competência os vários processadores disponíveis, a exemplo de sistemas operacionais e sistemas gerenciadores de bancos de dados já disponíveis e também suas futuras gerações. Essa demanda, portanto, precisa de uma resposta rápida da academia, que precisará rever seus critérios quanto à adoção de linguagens concorrentes e à disseminação dessa filosofia de programação entre seus alunos. É importante salientar que qualquer mudança de rumo adotada pela academia só deverá alcançar efeito prático após alguns anos, quando da efetiva inserção do profissional no mundo do trabalho.

2.5.2. Sobre os aspectos gerais do ensino de programação

A seguir apresentaremos os resultados obtidos a partir das 34 respostas da quarta parte do formulário, onde os respondentes expressam suas opiniões sobre algumas afirmativas colocadas a respeito do ensino de programação.

No formulário, as respostas possíveis eram: "CONCORDO FORTEMENTE", "CONCORDO", "NÃO SEI", "DISCORDO", "DISCORDO FORTEMENTE". Para efeito de tabulação, as respostas "CONCORDO FORTEMENTE" e "CONCORDO" foram fundidas na classe "CONCORDO" e as respostas "DISCORDO" e "DISCORDO FORTEMENTE" foram fundidas na classe "DISCORDO". Sendo assim, discutiremos adiante as respostas considerando as classes "CONCORDO", "DISCORDO", "NÃO SEI" e "NÃO RESPONDEU".

Na concepção original do formulário, apenas as respostas "DISCORDO", "CONCORDO" e "NÃO SEI" estavam presentes. A extensão da resposta "DISCORDO" para "DISCORDO FORTEMENTE" e de "CONCORDO" para "CONCORDO FORTEMENTE" teve como objetivo ampliar a escala de respostas e oferecer opções graduais para o participante. Essa técnica, conhecida como "escala de intensidade" é um artifício usual em pesquisas de opinião, visando evitar a tendência de posicionamento no grau intermediário, o que ocorre muito comumente com escalas de três graus [Marconi-90]. A fusão de classes, portanto, é o processo inverso, visando a normalização das respostas para fins de tabulação.

Para obter um valor único para cada afirmativa desprezamos as respostas da classe "NÃO RESPONDEU", e consideramos as respostas da classe "CONCORDO" como positivas, as da classe "DISCORDO" como negativas e as da classe "NÃO SEI" como nulas, para que as classes eliminem-se entre si, resultando na diferença entre as respostas. O resultado é dividido pelo número total de respostas, obtendo-se um valor percentual normalizado entre o intervalo (-100%, +100%), que mede o fator de concordância para cada afirmativa colocada. A fórmula usada é a seguinte:

$FC_i = 100\frac{(C_i - D_i)}{(C_i + D_i + N_i)}$, onde:	**FC_i** = Fator de concordância para a i-ésima afirmativa **C_i** = número de respostas "CONCORDO" para a i-ésima afirmativa **D_i** = número de respostas "DISCORDO" para a i-ésima afirmativa **N_i** = número de respostas "NÃO SEI" para a i-ésima afirmativa

Equação 1 – Cálculo do Fator de Concordância com as opiniões sobre as afirmativas ligadas ao ensino de programação

A Tabela 2 apresenta os resultados já normalizados. Os mesmos resultados são apresentados graficamente na Figura 7.

#	AFIRMATIVA	RESPOSTAS DISCORDO	RESPOSTAS CONCORDO	RESPOSTAS NÃO SEI	FATOR DE CONCORDÂNCIA
A	É necessário estabelecer uma separação entre o ensino de programação e o ensino de linguagens de programação para o alcance de melhores resultados	42,4%	51,5%	6,1%	9,1%
B	As disciplinas ligadas ao ensino de programação carecem de uma padronização de conteúdos nos cursos de graduação de seu conhecimento	23,5%	53,0%	23,5%	29,4%
C	A seqüência na qual os conteúdos das disciplinas ligadas ao ensino de programação são ministrados é importante para obtenção de melhores resultados	0,0%	91,2%	8,8%	91,2%
D	A seqüência na qual os conteúdos das disciplinas ligadas ao ensino de programação são ministrados varia nos cursos de graduação de seu conhecimento	5,9%	76,5%	17,6%	70,6%
E	É necessário que o ensino de programação seja menos dependente dos paradigmas de programação, para formar profissionais mais adaptáveis às evoluções que ocorrem na área	57,6%	30,3%	12,1%	-27,3%
F	Depois de graduados, os profissionais no mercado conseguem se adaptar com facilidade às mudanças e evoluções que ocorrem na área de programação	43,8%	37,5%	18,8%	-6,3%
G	A parte prática do ensino de programação é mais importante do que a parte teórica	79,5%	17,6%	2,9%	-61,8%
H	A parte prática e a parte teórica do ensino de programação são igualmente importantes	20,6%	76,5%	2,9%	55,9%
I	As ferramentas de ensino auxiliado por computador (tutoriais de software) trazem benefícios ao aprendizado de programação, quando bem aplicadas	9,1%	57,6%	33,3%	48,5%

Tabela 2 – Opinião dos professores sobre as afirmativas ligadas ao ensino de programação

O objetivo da afirmação **A** é testar a hipótese de que o enfoque das disciplinas de programação deve ser "ensinar programação" ao invés de "ensinar uma linguagem de programação". No primeiro caso, uma ênfase maior é dada aos conceitos formais como técnicas de programação, projeto e análise de algoritmos, exploração de estruturas lógicas, fluxo de controle, estruturas de dados, teoria, abstração, domínios de problemas. A linguagem de programação faz parte do processo como uma ferramenta para instanciação dos conceitos. Já no segundo caso, enfatiza-se mais os aspectos ligados à implementação da linguagem e à idéia de como traduzir os conceitos abstratos para uma linguagem formal.

Em certos casos o ensino de uma linguagem de programação é simplesmente o estudo da sintaxe e da filosofia de uma linguagem.

Como podemos perceber, o resultado obtido para esta afirmação não foi claro. Há uma ligeira inclinação à confirmação da hipótese (fator de concordância = 9,1%), o que sugere que este assunto goza de uma certa polêmica entre a comunidade docente.

A assertiva **B** se refere à uniformização dos conteúdos curriculares das disciplinas ligadas à área de programação nos cursos de graduação. Na opinião de 53,0% dos respondentes, os cursos de graduação apresentam diferenças de conteúdo e apenas 23,5% acreditam que não há variação perceptível. O fator de concordância de 29,4% pode indicar que não há um razoável entendimento entre as instituições de ensino quanto ao *que* se deve ensinar aos alunos de graduação na área de programação.

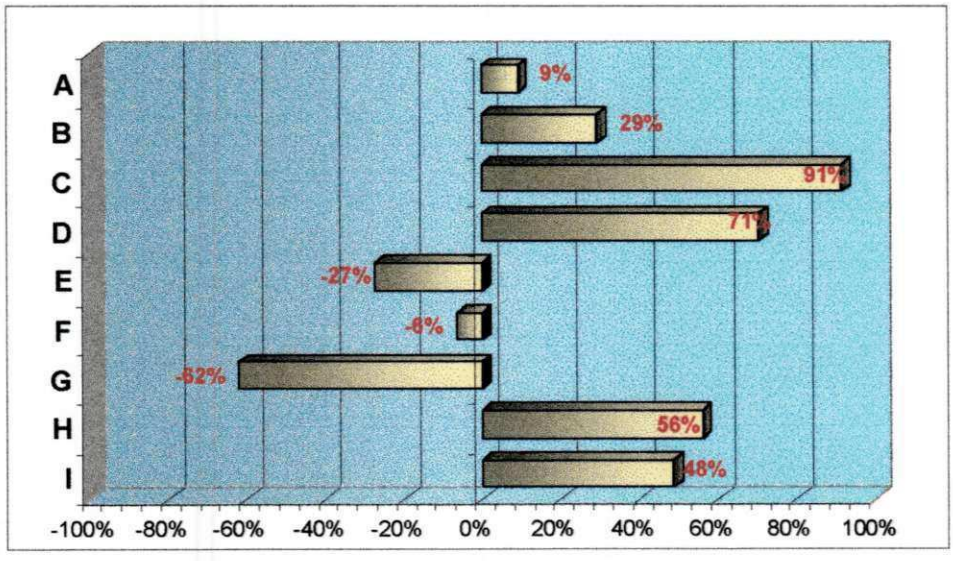

Figura 7 – Opinião dos professores sobre as afirmativas ligadas ao ensino de programação (representação da Tabela 2)

Em relação à assertiva **C** (sobre o quanto a seqüência na qual os conteúdos das disciplinas ligadas ao ensino de programação são ministrados é importante para obtenção de melhores resultados), o grau de concordância foi de 91,2%, apresentando uma confirmação do que nos parece, antes de qualquer análise, óbvio. Entretanto, os resultados da assertiva **D** (sobre a divergência na seqüência na qual os conteúdos são ministrados) obteve um fator de concordância de 70,6%, o que nos aponta que os professores concordam que há diferenças quanto à seqüência na qual os conteúdos são apresentados aos alunos.

Embora os resultados sejam baseados "no que os professores conhecem" e não necessariamente no que foi observado diretamente nos currículos dos cursos de graduação, eles são um indicativo de que há discrepâncias, propositais ou não, no entendimento

coletivo sobre qual a ordem em que devemos ensinar os conceitos ligados à área de programação.

A assertiva **E** tem como objetivo testar a hipótese de que a dependência de paradigmas no ensino de programação contribui para uma possível dificuldade de adaptação dos profissionais às evoluções que ocorrem na área de programação. A hipótese partiu da observação do fato de que os anos 90 têm se caracterizado por um forte crescimento e popularidade do paradigma de orientação a objetos [Bruce-96], e uma conseqüente necessidade de adaptação dos profissionais graduados anteriormente a esse período aos novos conceitos desse paradigma [Whitelaw-95]. Na opinião da maioria dos professores, não é necessário diminuir a dependência de paradigma no ensino de programação. Neste caso, podemos concluir que, na opinião dos professores, a adaptabilidade do profissional deve estar ligada a outro fator aqui não mencionado.

O resultado para a afirmação **F** (sobre a adaptabilidade dos profissionais após a graduação) revela que a maioria dos professores (57,6%) *não* acredita que os profissionais graduados terão facilidade de adaptação às mudanças que ocorrem na área de programação. Já os resultados para as afirmações **G** e **H** mostram que há um consenso entre os professores de que a teoria e a prática são igualmente importantes no ensino de programação.

Quanto à afirmação **I** (sobre a aplicabilidade de tutoriais de software no ensino de programação), constatamos que grande parte dos professores acredita nos benefícios do uso de tutoriais de software para o ensino de programação. Nesta afirmação, o índice de professores que "não sabem" foi alto (o maior entre todas as afirmativas), o que é um indicativo de que as ferramentas de software para o ensino são pouco usadas ou, quando o são, seus benefícios não são avaliados.

2.6. Considerações sobre os resultados da pesquisa

A análise dos comentários livres dos respondentes (quinta parte) sobre a pesquisa revelou que, em muitos casos, o respondente buscava justificar alguma resposta fornecida no formulário. Os comentários livres também foram importantes para coletarmos relatos sobre algumas experiências individuais relacionadas ao ensino de programação. A seguir, apresentamos uma discussão baseada nas observações fornecidas.

Com relação à linguagem de programação usada nas disciplinas, observamos que em alguns cursos esta é uma escolha do professor que ministra a disciplina. Mesmo que essa prática tenha se verificado em uma pequena minoria de cursos, isso sugere que a linguagem de programação não é importante, uma vez que qualquer uma pode ser escolhida.

Quanto ao que foi discutido na seção "2.5.2. Sobre os aspectos gerais do ensino de programação" (item A), observamos um equilíbrio de opiniões quanto à necessidade de estabelecer uma maior separação entre o ensino de programação e o ensino de linguagens de programação para o alcance de melhores resultados. Entretanto, muitos comentários foram colocados em relação a esta questão.

Grande parte dos comentários gira em torno da definição simplificada de que ensinar programação significa ensinar a resolver problemas usando-se uma ferramenta – a linguagem de programação. Sendo assim, a implementação da solução dependerá da ferramenta disponível e a qualidade da implementação dependerá do domínio sobre a ferramenta. Em torno disso, cada respondente busca uma justificativa para seu posicionamento quanto à questão e, a partir delas, podemos extrair as seguintes conclusões:

A. Aqueles que concordam com uma maior separação (51,5%), acreditam que é preciso tornar um pouco mais visível o que é a técnica de solução de problemas e o que é a ferramenta usada para a solução do problema. Para eles essa diferenciação de conceitos é uma questão de enfoque do professor e não, necessariamente, de estabelecer uma disciplina para cada conceito.

B. Aqueles que discordam da separação entre o ensino de programação e a linguagem de programação (42,4%), acreditam que as formas de resolver o problema proposto dependem fundamentalmente da ferramenta disponível e que, sem o domínio da ferramenta é difícil absorver as técnicas de solução de problemas. Portanto, é impossível dissociar uma coisa da outra.

Os comentários livres também foram importantes para coletarmos alguns relatos sobre o uso de paradigmas no ensino introdutório de programação. Pôde-se perceber uma certa consciência quanto a predominância do paradigma imperativo nas disciplinas introdutórias de programação e alguma preocupação com a necessidade de realização de experiências com outros paradigmas nessa fase, em função das dificuldades que os alunos apresentam para mudar para um segundo paradigma após aprender inicialmente o paradigma imperativo. A partir disso, surgiram relatos sobre algumas experiências individuais relacionadas ao uso de um ou outro paradigma de programação, em opção ao paradigma imperativo. Alguns relatam que a adoção de uma linguagem orientada a objetos tem trazido benefícios à aprendizagem dos alunos, em função de uma maior facilidade para modelagem de problemas reais e do mapeamento direto entre os objetos e as entidades do mundo real.

Alguns professores citaram experiências com linguagens funcionais, adotando-as já nas primeiras disciplinas de programação. Esses professores alegaram ter tido mais facilidade

em ensinar alguns conceitos, tais como escopo, recursividade e ativação de funções. Este tipo de experiência mostra que, a despeito do total domínio do paradigma imperativo nas primeiras disciplinas de programação, alguns resultados positivos podem ser também alcançados com o uso de abordagens alternativas.

Pelo que observamos, no geral, a avaliação de alternativas às linguagens imperativas para o ensino introdutório de programação parece ser uma tendência nos currículos, tanto no País quanto no exterior. O CSAC/CSAB – *Computer Science Accreditation Commission of the Computing Sciences Accreditation Board*, uma organização de certificação e avaliação de cursos de computação nos Estados Unidos (reconhecida pelo Departamento de Educação e pelo Conselho de Educação Superior locais), tem realizado anualmente entre as universidades americanas uma pesquisa que envolve questões sobre os docentes, departamentos, alunos e sobre o currículo dos cursos oferecidos. O relatório anual da pesquisa feita em 1995, aponta que o principal paradigma ensinado na ocasião era o *imperativo* para 73% das instituições e era o *objeto* para 36% (segundo o autor da pesquisa, a soma ultrapassa 100% pelo fato de algumas instituições terem informado que ambos os paradigmas eram igualmente principais). No ano de 1996, estes números mudaram para 54% e 55%, respectivamente, demonstrando a preferência decrescente pelo paradigma imperativo, com o passar do tempo, para o ensino de programação durante o curso [McCauley-96].

Analisando esta informação sob o ponto de vista das linguagens de programação usadas no ensino introdutório de programação nos EUA, observamos a presença cada vez menor das linguagens imperativas (ex. C, Pascal, FORTRAN) e a crescente presença de linguagens de outros paradigmas, tais como orientação a objeto (C++, Eiffel, Java), funcional (Scheme) e concorrente (Ada), conforme podemos ver na Tabela 3.

LINGUAGEM DE PROGRAMAÇÃO	PRIMEIRA LINGUAGEM DE PROGRAMAÇÃO ENSINADA			
	EM 1995	EM 1996	EM 1997	PRETENSÃO DE MUDAR PARA
Pascal	**36%**	23%	6%	-
C++	32%	**39%**	**47%**	4%
C	17%	14%	11%	2%
Ada	12%	18%	19%	2%
Eiffel	2%	-	2%	-
FORTRAN	2%	-	-	-
Scheme	2%	4%	4%	-
Visual Basic	-	2%	-	-
Modula 2	-	2%	-	-
Java	-	-	8%	11%

Tabela 3 – Linguagens ensinadas em primeiro lugar em cursos de computação nos EUA (fonte: CSAC/CSAB [McCauley-96])

Além da preferência ascendente ao longo do tempo, demonstra-se que as linguagens C++, Ada e Java irão crescer ainda mais nos currículos, a julgar pelos números mostrados na coluna "PRETENSÃO DE MUDAR PARA", da Tabela 3, que indicam o percentual de instituições que *não* usam a linguagem em questão, mas pretendem mudar para ela.

Segundo Allen Tucker, a metodologia de ensino que introduz os conceitos de programação aos milhares de estudantes a cada ano é dominada por uma coleção de linguagens de programação freqüentemente ultrapassadas, onde predomina a solução de pequenos problemas, ausência de trabalho em equipe e ausência de métodos formais [Tucker-96a]. Isto pôde em parte ser comprovado pela pesquisa 1, que mostra que algumas linguagens como FORTRAN e COBOL ainda são ensinadas em cursos de graduação, sendo até mais usadas como linguagens introdutórias do que C++, Delphi/Pascal (OOP), Visual Basic e Java, por exemplo.

Por ter sido realizada pela primeira vez, nossa pesquisa sobre o ensino de programação nas IES brasileiras não contabiliza informações relativas aos anos anteriores, o que não nos permite estabelecer comparações nesse nível. Entretanto, percebe-se nos comentários dos participantes e também em levantamentos informais, a gestação de propostas de mudanças na forma de ensinar programação e na escolha de linguagens e paradigmas, bem como de avaliar a eficácia das mudanças pioneiras já realizadas por outras instituições nessa direção, principalmente na adoção do paradigma "objeto" em substituição ao "imperativo".

De fato, há evidências de que a adoção do paradigma "objeto" para o ensino introdutório de programação seja uma tendência mundial. Um exemplo disso é o crescimento na adoção de linguagens como C++ e Java como primeira linguagem em cursos da área de computação nos EUA (vide Tabela 3) e da vasta literatura já disponível para o uso de orientação a objetos no ensino introdutório de programação [Sharp-96], [Astrachan-96b], [Kereki-97].

Uma experiência prática nessa direção, por exemplo, foi feita pela *Universidad ORT Uruguay*, trocando-se a linguagem Pascal pela linguagem Smalltalk na primeira disciplina de programação. A mudança foi acompanhada de um plano de avaliação e da adoção de algumas ferramentas didáticas adicionais (ex. laboratórios com uma TV conectada ao computador do professor visando a difusão para os alunos das ações do professor durante a aula). Além disso, os alunos ficavam mais concentrados na aula uma vez que, ao final de cada aula, poderiam ter acesso a todas as anotações, programas fonte, observações do professor e guia de exercícios. A avaliação após um ano mostrou um aumento na aprovação, redução de evasão e aumento da qualidade dos trabalhos dos alunos [Kereki-97]. Observe-se, entretanto, a mudança na metodologia como um importante aliado na mudança da linguagem usada, para o alcance de melhores resultados.

3. Pesquisa 2: A Prática do Profissional de Programação

3.1. Apresentação

A realidade vivida pelo profissional certamente é um assunto de grande interesse para a academia, tendo em vista a relação de interdependência que trava com o mercado de trabalho. Com relação aos profissionais de programação, especificamente, a sintonia entre os currículos de graduação e a prática do profissional após sua formação é um fator que pode influenciar definitivamente em sua absorção por parte do mercado e em sua empregabilidade.

Por outro lado, sabemos que o excessivo direcionamento dos currículos para o mercado pode significar "queimar etapas" no processo educacional, uma vez que a abordagem de tópicos ligados à conceituação, fundamentação teórica e formalização matemática são preteridos em favor de abordagens mais práticas, imediatistas e, em muitos casos, vinculadas a produtos de penetração mercadológica momentânea.

Portanto, é conveniente que o ensino de programação busque um equilíbrio entre a prática pedagógica e a real necessidade do profissional no mercado de trabalho, sem sacrifício para nenhuma das partes. Mas, afinal, como é a realidade atual do profissional de programação no mercado de trabalho? É possível continuar aprendendo a despeito das evoluções na área de programação? Que mecanismos são usados pelos egressos para se manterem atualizados?

Com o intuito de conhecer algumas respostas e conhecer a realidade do profissional de programação após sua formação, elaboramos uma pesquisa denominada "Coleta de informações sobre a prática dos profissionais de programação", consistindo de um formulário com questões sobre o assunto. O formulário foi transcrito para o formato HTML e publicado com livre acesso na *Web*, em um servidor HTTP do DSC/UFPb. Uma cópia dos formulários nas línguas portuguesa e inglesa podem ser vistas nos anexos C e D, respectivamente.

3.2. Materiais e métodos

O mecanismo utilizado para obtenção das informações foi um formulário, composto por seis partes. Na primeira parte, solicitávamos ao respondente que fornecesse as seguintes informações (veja Figura 8):

- Função que exerce atualmente
- Nome do curso de graduação

- Ano da graduação
- Universidade ou Instituição de ensino
- País onde situa-se a universidade ou Instituição de ensino
- Experiência com programação (em anos)

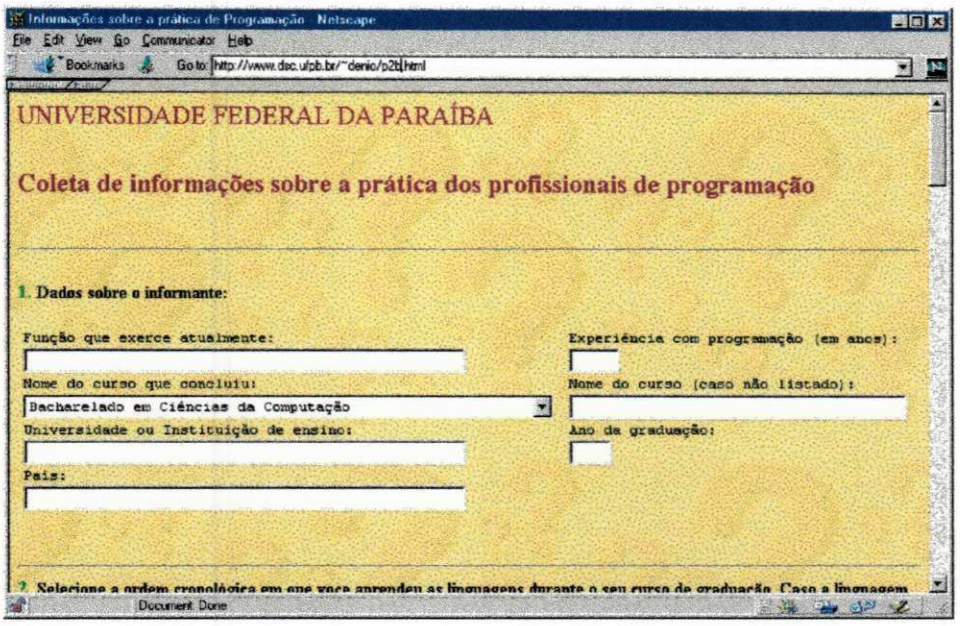

Figura 8 – Primeira parte do formulário eletrônico da Pesquisa 2: dados do respondente

Na segunda parte inquiríamos o profissional sobre as linguagens que aprendeu na graduação e a ordem cronológica em que foram aprendidas (veja a Figura 9). As linguagens eram dispostas em uma lista selecionável (conhecida como *combo* ou *select list*) contendo 86 linguagens de programação conhecidas. O respondente poderia escolher apenas uma delas em cada um dos espaços reservados para "primeiro lugar", "segundo lugar", "terceiro lugar" e "quarto lugar". Nos casos em que a linguagem em particular não constasse na lista o respondente poderia fornecer o nome no campo reservado para "Outra".

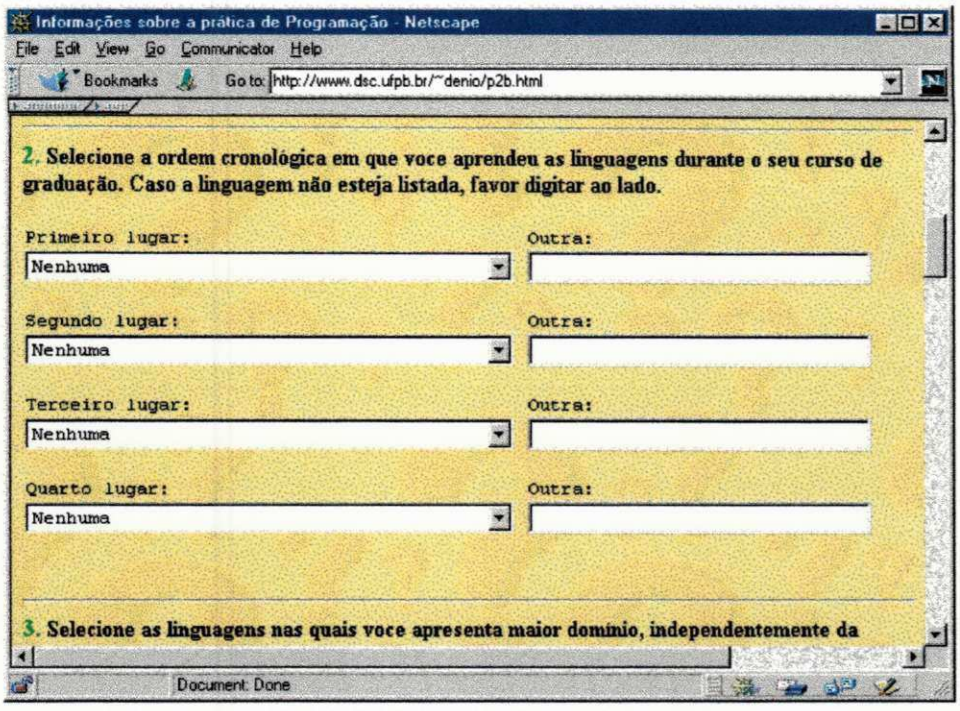

Figura 9 – Segunda parte do formulário eletrônico da Pesquisa 2: linguagens de programação aprendidas

Na terceira parte, perguntávamos ao profissional quais eram as linguagens de programação em que ele apresentava maior domínio. Até quatro linguagens poderiam ser informadas, na ordem do maior para o menor domínio. As linguagens eram colocadas em listas de seleção e a mecânica de funcionamento era a mesma da segunda parte. A terceira parte do formulário pode ser vista na Figura 10.

Ao perguntarmos aos profissionais sobre as linguagens que aprendeu (parte 2) e as linguagens que mais domina (parte 3), estávamos considerando que o profissional poderia adotar outras ferramentas de programação ao longo de sua prática profissional no mercado de trabalho, além daquelas conhecidas na graduação.

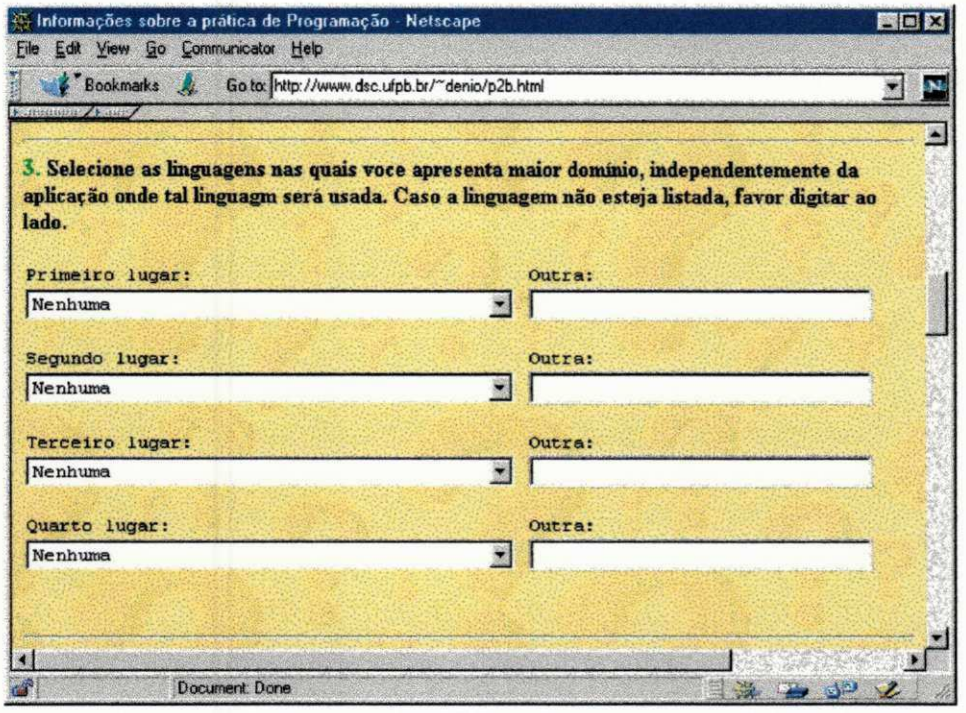

Figura 10 – Terceira parte do formulário eletrônico da Pesquisa 2: linguagens de programação de maior domínio

Na quarta parte questionávamos o profissional sobre qual o grau de dificuldade que ele tinha para se manter atualizado quanto às evoluções das técnicas de programação desde a conclusão da sua graduação. A motivação para este questionamento era obter uma idéia sobre quão preparado está o profissional para acompanhar as evoluções do dia-a-dia na área de programação. Para a pergunta apresentada, as respostas oferecidas eram: "GRANDE", "RAZOÁVEL", "POUCA" ou "NENHUMA".

A quinta parte visava obter do profissional informações sobre os mecanismos mais freqüentemente usados por ele para se manter atualizado na área. Algumas opções foram apresentadas e o profissional foi convidado a informar com que freqüência se utilizava daquela opção em particular, usando a seguinte escala para resposta: "NUNCA", "QUASE NUNCA", "ÀS VEZES", "QUASE SEMPRE" ou "SEMPRE". Nos casos em que o profissional se utilizasse de algum outro mecanismo de atualização não apresentado no formulário, ele poderia informar livremente nos espaços reservados para "Outro", usando a mesma escala de respostas das opções pré-definidas. A quarta e quinta partes do formulário podem ser vistas na Figura 11.

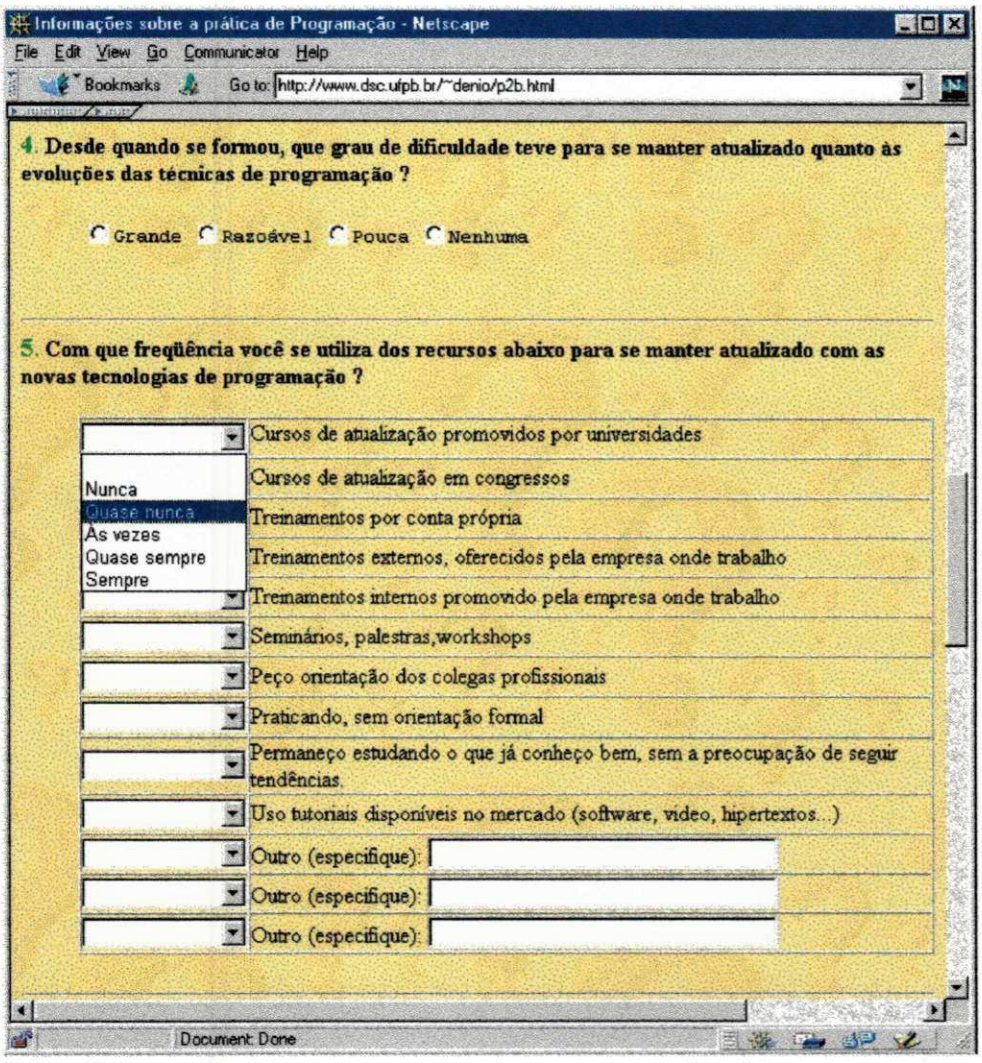

Figura 11 – Quarta e quinta partes do formulário eletrônico da Pesquisa 2: grau de dificuldade de atualização e os mecanismos usados para atualização

A sexta parte colocava a seguinte questão: *"Que grau de dificuldade você tem ao aprender uma nova linguagem ou se adaptar a uma inovação ligada à prática da programação?"*. As respostas oferecidas eram: "GRANDE", "RAZOÁVEL", "POUCA" ou "NENHUMA". Em seguida, tentávamos identificar o motivo da resposta apresentando algumas opções e convidando o profissional a opinar sobre a influência daquela opção em particular sobre o grau de dificuldade informado. Para cada opção, a opinião do profissional era expressa usando a seguinte escala: "DE FORMA ALGUMA", "PROVAVELMENTE NÃO", "TALVEZ", "PROVAVELMENTE SIM" ou "CERTAMENTE". Nos casos em que o profissional acreditasse haver algum outro motivo para o seu grau de dificuldade declarado, o qual não constasse do formulário, ele poderia informar livremente nos espaços reservados para "Outro", usando a mesma escala de respostas das opções pré-definidas. A sexta parte pode ser vista na Figura 12.

Cabe aqui um comentário sobre a diferença entre a questão 4 e a questão 6. A questão 4 é mais ampla e questiona o respondente quanto a sua dificuldade de se manter atualizado

frente às evoluções genéricas que ocorrem em conceitos e técnicas de programação. No caso da questão 6, buscam-se informações específicas sobre a dificuldade de aprender uma nova linguagem de programação e, portanto, aborda a dificuldade para o aprendizado de novas ferramentas. Uma maior discussão sobre as diferenças entre a cultura da programação e as ferramentas de programação será feita adiante na seção *4.3.4.*

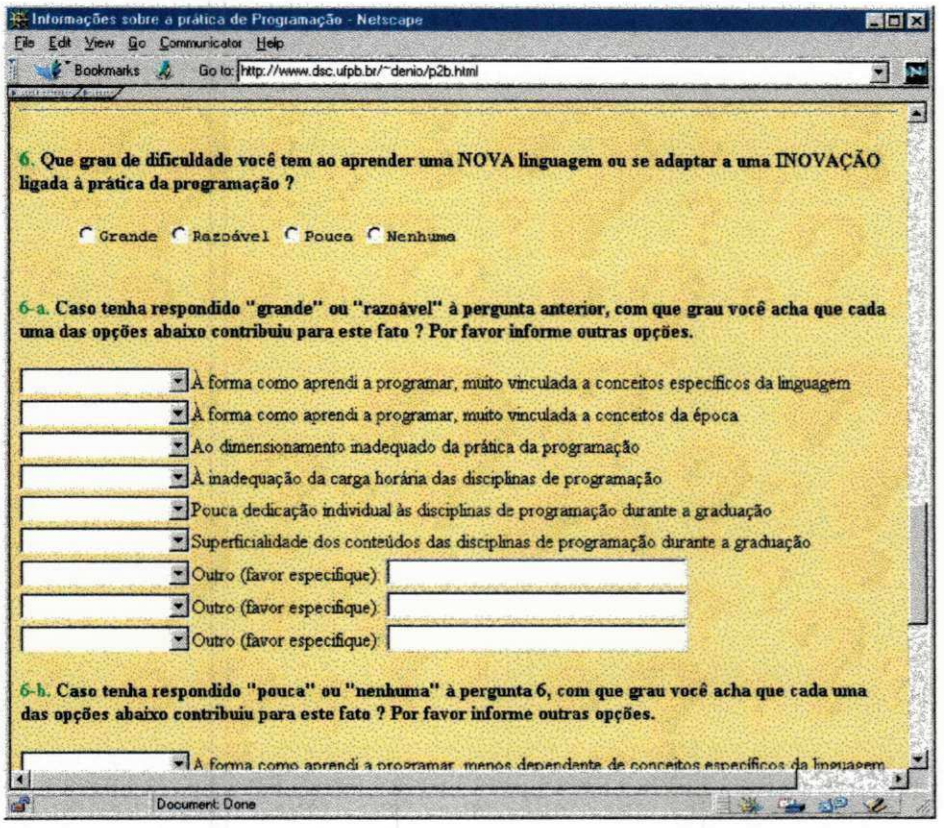

Figura 12 – Sexta parte do formulário eletrônico da Pesquisa 2: grau de dificuldade de para aprender uma nova linguagem de programação e os prováveis motivos para os que têm grande ou razoável dificuldade

Além de procurar conhecer o grau de dificuldade para a questão apresentada na sexta parte, procuramos investigar também o provável motivo para o grau de dificuldade declarado. Para tanto, desdobramos a seqüência da sexta parte em duas: **6.a** e **6.b**. Caso o respondente declarasse ter grande ou razoável dificuldade, ele seguiria o formulário pela questão **6.a**. Caso declarasse ter pouca ou nenhuma dificuldade ele seguiria pela questão **6.b**. A existência dessa bifurcação no sequenciamento da sexta parte se deve ao fato de que as opções previamente oferecidas no formulário como razões prováveis para um baixo grau e para um alto grau de dificuldade eram diferentes. Os desdobramentos **6.a** e **6.b** podem ser vistos na Figura 12 e na Figura 13, respectivamente.

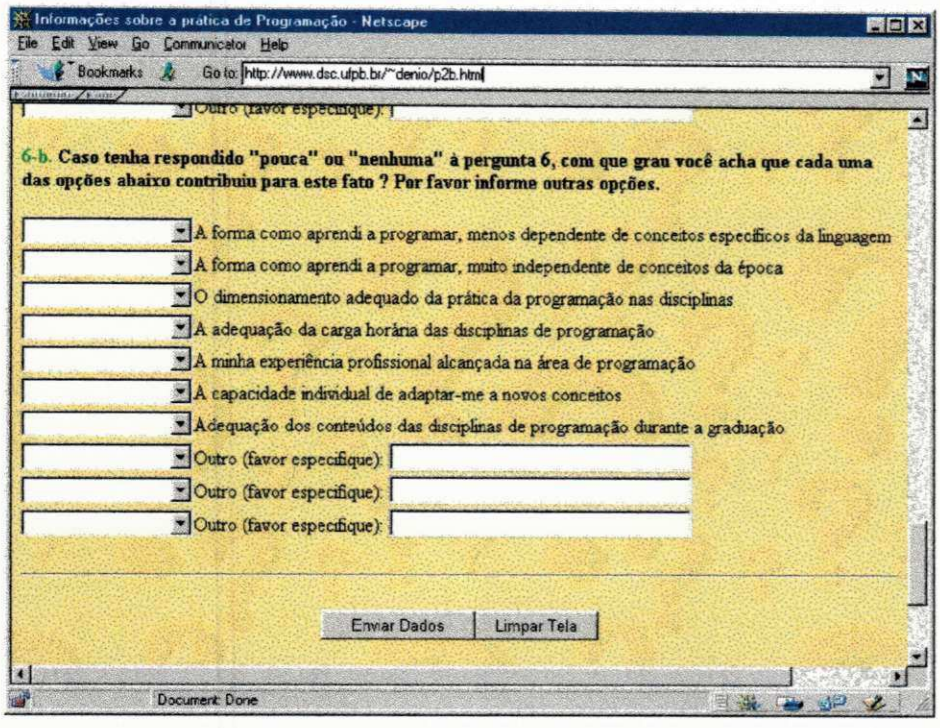

Figura 13 – Sexta parte do formulário eletrônico da Pesquisa 2: grau de dificuldade de para aprender uma nova linguagem de programação e os prováveis motivos para os que têm pouca ou nehuma dificuldade

3.3. O público alvo e a divulgação

Nesta pesquisa, nossa população é composta por profissionais graduados, de qualquer área, que desenvolvem alguma atividade ligada à pratica da programação. Ao buscarmos as opiniões de profissionais egressos de "qualquer área", entendemos estar estendendo a pesquisa para aqueles cursos que oferecem disciplinas de informática ligadas à programação, como nos cursos de Engenharia Elétrica, Engenharia Civil, Matemática, entre outros, em cujas profissões já se verifica elevada interdisciplinaridade com a informática e razoável grau de importância do conhecimento de programação de computadores.

A divulgação desta pesquisa foi feita no meio profissional através de correio eletrônico, usando listas de discussão sobre programação, grupos de discussão sobre ferramentas de programação e publicação da *home page* da pesquisa em *sites* de pesquisa e busca[1]. Também foram enviados convites diretamente para *software houses* brasileiras e internacionais e diretamente para profissionais, através de correio eletrônico.

Além de publicarmos o formulário de forma eletrônica, a pesquisa também foi realizada diretamente junto a algumas empresas das cidades de João Pessoa e Campina Grande,

[1] A home page das pesquisas poderia ser acessada diretamente no URL http://www.dsc.ufpb.br/~denio/pesquisa.html ou pesquisada nos seguintes endereços: Altavista (http://altavista.digital.com), Yahoo (http://www.yahoo.com), Infoseek (http://www.infoseek.com), Webcrawler (http://www.webcrawler.com), Cade? (http://www.cade.com.br) e Surf (http://www.surf.com.br).

ambas na Paraíba, e em Recife, Pernambuco. Nesses casos, buscamos os profissionais da área de informática e suas opiniões foram registradas na versão impressa do formulário, sendo depois transcritas para o formulário eletrônico da WWW.

Ao todo foram enviados 6.430 convites na forma de mensagens de correio eletrônico diretamente para profissionais e instituições diferentes (desprezando-se nesta contabilidade as mensagens indiretas produzidas pelas listas de discussão), no período compreendido entre os meses de fevereiro e julho de 1997.

3.4. Resultados

A pesquisa obteve respostas de 439 respondentes, sendo 376 (86%) oriundas do Brasil, 63 (14%) de outros países. Considerando os continentes de onde as respostas foram oriundas, tivemos a seguinte distribuição: América do Sul (87,7%), América do Norte (5,9%), Europa (5,0%), Ásia (0,9%), Oceania (0,2%), Africa (0,2%).

Dentre todas as respostas, 93% delas foram obtidas via formulário eletrônico através da WWW, e os 7% restantes obtidos diretamente pelo formulário impresso.

A experiência com programação declarada foi em média 9,4 anos (desvio padrão de 6,4; min=0, máx=33) e o tempo médio decorrido desde o fim da graduação foi de 6,9 anos (desvio padrão de 6,8; min=0 máx=41). A distribuição dos respondentes quanto a sua formação básica se deu da seguinte forma: Bacharelado em Ciências da Computação (45%), Processamento de Dados (12%), Engenharia Elétrica (11%), Bacharelado em Informática (6%), Engenharia de Computação (4%), Matemática (3%), Engenharia Civil (3%) e outros cursos (16%). Esta distribuição pode ser vista na Figura 14.

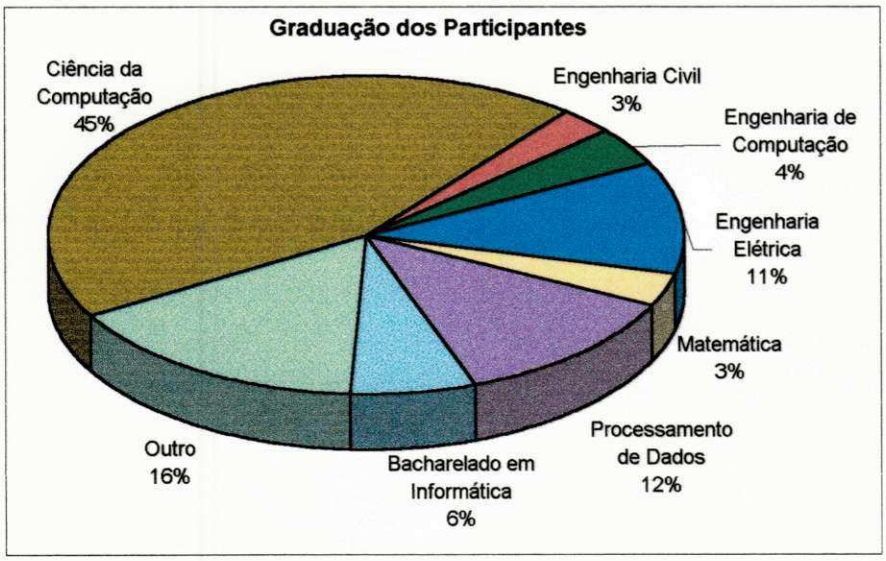

Figura 14 – Distribuição dos participantes da pesquisa pelo seu curso de graduação

Com relação à função exercida pelo respondente, observamos uma ampla distribuição. Uma vez que a função era informada em um campo de texto livre, tivemos respostas variadas e em, alguns casos, mais de uma função exercida simultaneamente, abrangendo inclusive parte do segmento acadêmico. Sendo assim, surgiram, por exemplo, respostas do tipo: "professor e analista de sistemas", "mestrando e consultor" ou "*software engineer and project manager*" (respondentes do formulário em inglês) etc. Nesses casos, as funções foram contabilizadas em todas as classes que apareciam na resposta, causando uma sobreposição percentual entre funções. Em decorrência disso, o total dos percentuais das funções ultrapassa 100%. A apuração das funções pode ser vista na Tabela 4.

FUNÇÃO	PARTICIPAÇÃO
Analista de sistemas	26%
Programador, desenvolvedor	16%
Aluno de pós-graduação (mestrado ou doutorado)	14%
Gerente, supervisor, líder ou coordenador de equipes de desenvolvimento	12%
Professor	11%
Analista de suporte	5%
Engenheiro de software	5%
Pesquisador	5%
Consultor ou assessor	4%
Administrador de redes e/ou sistemas multiusuários	3%
Outra	10%

Tabela 4 – Participação das funções ocupadas pelos respondentes da Pesquisa 2

3.4.1. Sobre o aprendizado de linguagens e paradigmas de programação

Na pesquisa, perguntamos aos profissionais quais as linguagens que eles aprenderam no curso de graduação, permitindo que informassem quatro linguagens, na ordem em que foram aprendidas.

A linguagem que mais apareceu como primeira linguagem aprendida foi Pascal (46,2%), seguida por FORTRAN (19,2%), BASIC (11,9%), Algol (4,8%), COBOL (3,0%), C (2,5%), Assembly (2,3%), PL/I (2,0%) e Clipper (1,0%). Outras linguagens somaram 3,8%, sendo que 3,3% dos respondentes não informaram a primeira linguagem que aprenderam.

A segunda linguagem aprendida pelos respondentes seguiu a seguinte ordem: C (19,8%), Pascal (16,5%), COBOL (14,7%), FORTRAN (6,3%), Assembly (4,8%), BASIC (4,1%), C++ (3,5%), Clipper (3,3%) e outras (19,2%). Entre os respondentes, 7,9% não indicaram uma segunda linguagem. Os valores percentuais e a ordem cronológica completa em que as linguagens de programação foram aprendidas podem ser vistos na Tabela 5.

LINGUAGEM	PARADIGMA PREDOMINANTE	APRENDIDO EM PRIMEIRO LUGAR	APRENDIDO EM SEGUNDO LUGAR	APRENDIDO EM TERCEIRO LUGAR	APRENDIDO EM QUARTO LUGAR
Pascal	Imperativo	46,2%	16,5%	6,4%	1,8%
FORTRAN	Imperativo	19,2%	6,3%	5,1%	2,3%
BASIC	Imperativo	11,9%	4,1%	1,3%	2,1%
Algol	Imperativo	4,8%	2,8%	1,0%	0,3%
COBOL	Imperativo	3,0%	14,7%	12,0%	6,5%
C	Imperativo	2,5%	19,8%	15,3%	10,4%
Assembly	Imperativo	2,3%	4,8%	8,9%	4,9%
PL/I	Imperativo	2,0%	2,0%	0,3%	1,0%
Clipper	Imperativo	1,0%	3,3%	6,1%	3,9%
C++	Objeto	0,5%	3,5%	9,2%	6,0%
Modula-2	Imperativo	0,5%	1,8%	1,0%	0,5%
X-Base	Imperativo	0,3%	2,8%	1,0%	1,8%
Prolog	Lógico	0,3%	2,5%	2,5%	5,7%
Smalltalk	Objeto	0,3%	0,3%	0,8%	1,0%
MUMPS	Imperativo	0,3%		0,8%	0,3%
Miranda	Funcional	0,3%		0,3%	
RPG	Imperativo	0,3%		0,3%	0,5%
Haskell	Funcional	0,3%			
Lisp	Funcional		2,0%	2,8%	4,2%
SQL	Imperativo		1,0%	3,8%	4,4%
Visual Basic	Imperativo		0,8%	0,8%	2,3%
Delphi OOP	Objeto		0,5%	1,0%	2,1%
Delphi	Imperativo		0,5%	0,5%	0,8%
Scheme	Funcional		0,3%	0,3%	
Logo	Imperativo		0,3%	0,3%	
Ada	Concorrente		0,3%		0,3%
Rexx	Imperativo		0,3%		
Oberon	Objeto		0,3%		
Perl	Imperativo			0,5%	0,3%
Java	Objeto			0,5%	3,6%
CSP	Concorrente			0,3%	0,5%
APL	Imperativo			0,3%	0,3%
Dataflex	Imperativo			0,3%	
JavaScript	Objeto			0,3%	0,3%
Simula	Objeto			0,3%	0,5%
Natural	Imperativo				0,5%
Tcl/Tk	Imperativo				0,3%
Unix Shell – sh, csh, ksh	Imperativo				0,5%
Outra		1,0%	1,0%	1,3%	1,3%
Não Respondeu		3,3%	7,9%	15,0%	28,8%

Tabela 5 – As linguagens de programação e a ordem cronológica em que foram aprendidas

Baseada na Tabela 5, a Figura 15 abaixo mostra uma representação gráfica das linguagens de programação que mais se destacaram como instrumento de aprendizado para os respondentes (valores percentuais arredondados). Visando melhor apresentação, os percentuais das disciplinas aprendidas em quarto lugar são omitidas, da mesma forma que a soma dos percentuais das linguagens não apresentadas é atribuída ao item "Outra".

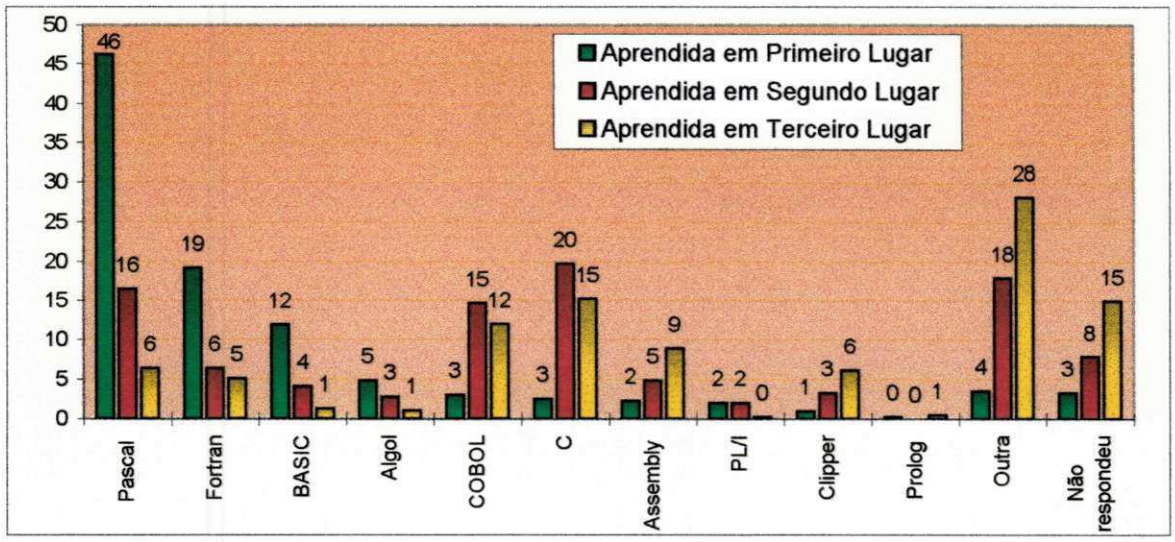

Figura 15 – Representação gráfica da ordem cronológica em que as linguagens de programação foram aprendidas (valores em percentual)

Computando-se todas as linguagens informadas e conhecendo-se o paradigma predominante de cada linguagem, podemos concluir que o primeiro contato com programação na graduação foi através do paradigma "imperativo" para 94% dos profissionais. O paradigma "objeto" foi o primeiro contato para apenas 1% dos profissionais, tendo o mesmo acontecido para o paradigma "funcional". O paradigma da segunda linguagem aprendida foi também o "imperativo" para 82% dos profissionais, o terceiro para 66% e o quarto para 46%. A Figura 16 demonstra a ordem em que os demais paradigmas foram aprendidos pelos profissionais.

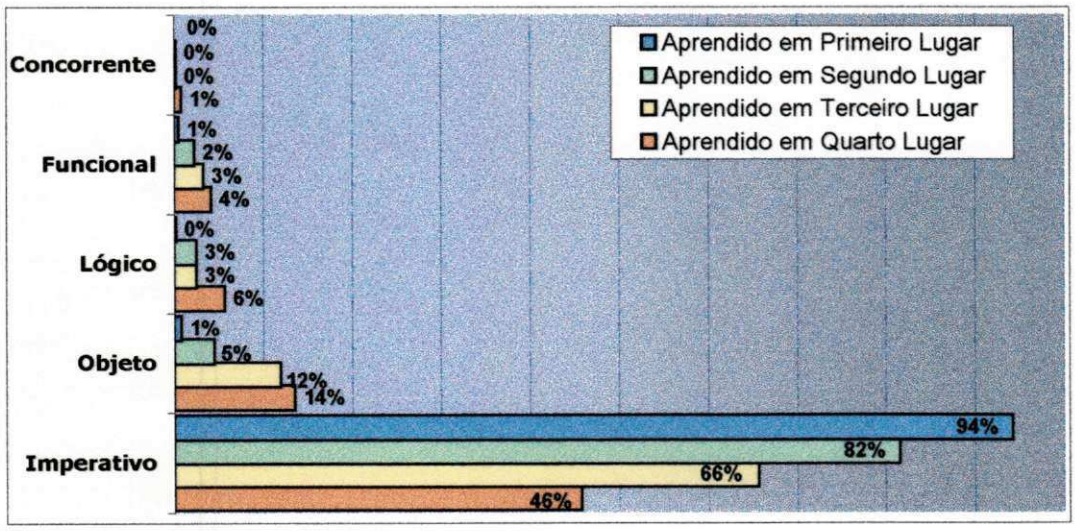

Figura 16 – Demonstrativo dos paradigmas e da ordem em que foram aprendidos pelos profissionais

A Figura 17 é baseada nos mesmos dados, mas nos mostra uma ligeira variação da Figura 16, a qual é centrada no paradigma. A Figura 17 é centrada na ordem cronológica em que

os paradigmas foram aprendidos pelos profissionais, permitindo uma melhor visualização sobre a seqüência de aprendizado dos paradigmas por parte dos profissionais.

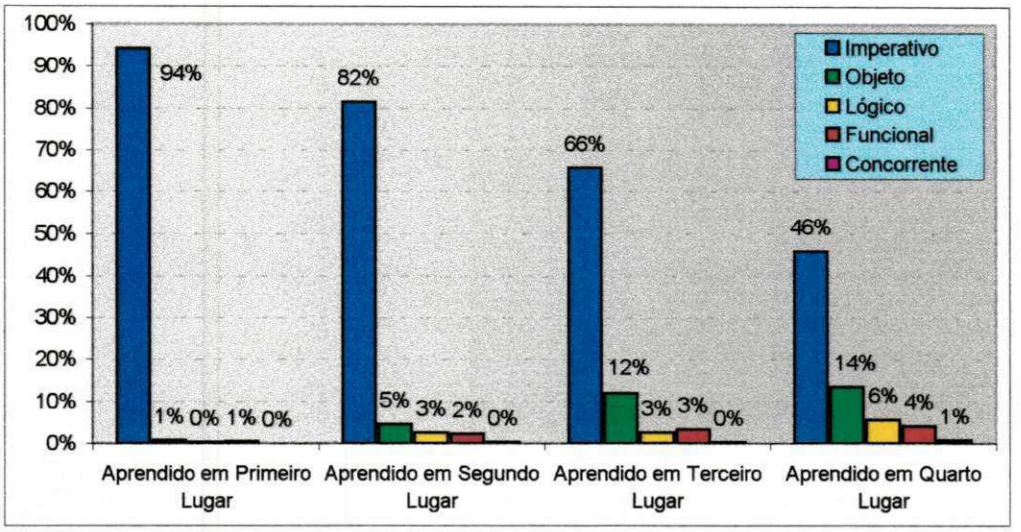

Figura 17 – Ordem cronológica do aprendizado dos paradigmas

3.4.2. Sobre o domínio de linguagens e paradigmas de programação

Na Pesquisa 2, perguntamos também aos profissionais quais as linguagens que eles mais dominam, permitindo que informassem quatro linguagens, na ordem de maior domínio. A linguagem que mais apareceu como primeira linguagem mais dominada foi C (26,0%), seguida por Clipper (10,3%), Pascal (10,0%), C++ (8,8%), Delphi (OOP) (6,8%), Visual Basic (4,8%), X-base (exceto Clipper) (3,5%), Assembly (3,4%), COBOL (3,2%). Todas as outras linguagens informadas somaram 19,7%, sendo que 3,5% não respondeu.

LINGUAGEM	PARADIGMA PREDOMINANTE	PRIMEIRO MAIOR DOMÍNIO	SEGUNDO MAIOR DOMÍNIO	TERCEIRO MAIOR DOMÍNIO	QUARTO MAIOR DOMÍNIO
C	Imperativo	26,0%	16,2%	6,7%	5,8%
Clipper	Imperativo	10,3%	7,4%	5,1%	4,2%
Pascal	Imperativo	10,0%	12,2%	14,6%	7,9%
C++	Objeto	8,8%	14,5%	7,8%	4,7%
Delphi OOP	Objeto	6,8%	3,6%	2,7%	3,4%
Visual Basic	Imperativo	4,8%	2,7%	4,0%	0,7%
X-Base (não Clipper)	Imperativo	3,5%	2,5%	1,3%	0,5%
Assembly	Imperativo	3,4%	1,6%	2,4%	1,3%
COBOL	Imperativo	3,2%	3,8%	2,4%	2,3%
SQL	Imperativo	2,4%	5,7%	4,5%	2,9%
BASIC	Imperativo	2,3%	6,5%	3,7%	3,4%
FORTRAN	Imperativo	2,1%	1,1%	1,8%	1,8%
Java	Objeto	1,9%	1,8%	5,1%	5,5%
Delphi	Imperativo	1,3%	0,5%	3,5%	2,9%
Natural	Imperativo	0,8%	0,8%		
MUMPS	Imperativo	0,8%	0,5%		

Smalltalk	Objeto	0,5%	0,2%	1,3%	0,5%
CSP	Concorrente	0,4%	0,5%	0,5%	0,2%
Pascal OOP	Objeto	0,4%	0,2%		0,2%
Prolog	Lógico	0,4%		1,3%	3,7%
Ada	Concorrente	0,4%		0,2%	
Perl	Imperativo	0,2%	1,6%	0,8%	1,3%
Unix Shell – sh, csh, ksh	Imperativo	0,2%	0,8%	0,2%	1,8%
Scheme	Funcional	0,2%		0,2%	
APL	Imperativo	0,2%		0,2%	
ML	Funcional	0,2%			
Eiffel	Objeto	0,2%			
JavaScript	Objeto	0,2%			0,2%
Lisp	Funcional		0,8%	2,4%	0,7%
PL/I	Imperativo		0,5%	1,0%	
Awk	Imperativo		0,2%	0,2%	
Rexx	Imperativo		0,2%	0,2%	0,2%
RPG	Imperativo		0,2%	0,2%	0,2%
Dataflex	Imperativo		0,2%		
Tcl/Tk	Imperativo		0,2%		
Dataflex OOP	Objeto		0,2%		
Simula	Objeto			0,8%	
Modula-2	Imperativo			0,5%	
Algol	Imperativo			0,2%	0,2%
Não Respondeu		3,5%	8,5%	22,4%	41,7%
Outra		4,6%	4,3%	1,8%	1,8%

Tabela 6 – As linguagens mais dominadas pelos respondentes da pesquisa

Baseada na Tabela 6, a Figura 18 abaixo mostra uma representação gráfica das linguagens de programação que mais se destacaram como instrumento de aprendizado para os respondentes (valores percentuais arredondados). Visando melhor apresentação, os percentuais das disciplinas aprendidas em quarto lugar são omitidas, da mesma forma que a soma dos percentuais das linguagens não apresentadas é atribuída ao item "Outra".

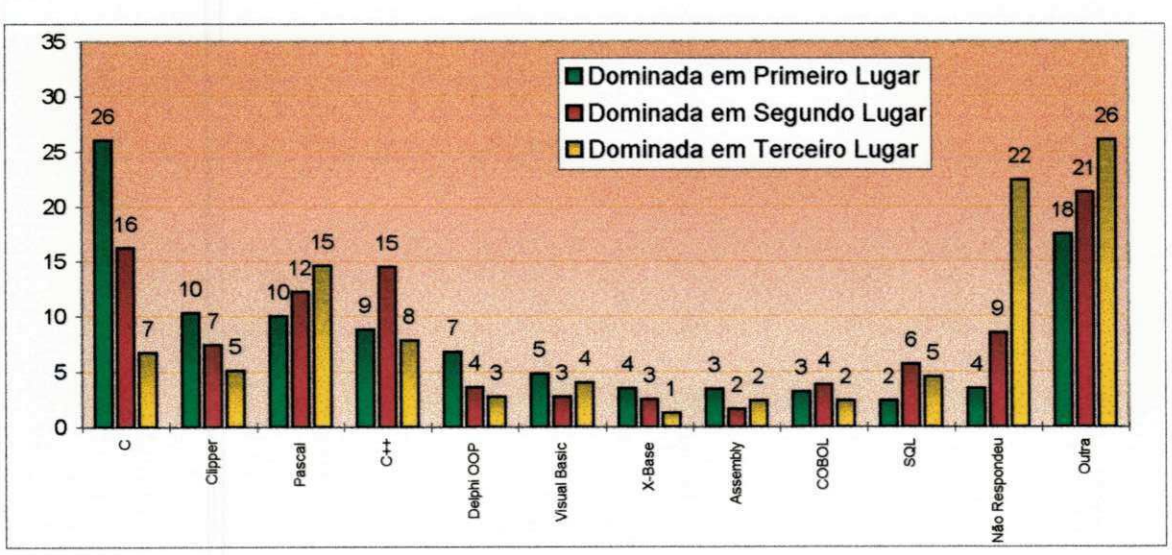

Figura 18 – As linguagens de programação mais dominadas pelos profissionais (visualização gráfica da Tabela 6)

Computando-se todas as linguagens informadas e conhecendo-se o paradigma predominante de cada linguagem, podemos concluir que o paradigma "imperativo" domina

amplamente o mercado de trabalho com 72% do total, em termos de conhecimento principal por parte do profissional. O paradigma "objeto" aparece em 19%, enquanto que o paradigma "concorrente" representa 1% das linguagens mais dominadas pelos profissionais. Considerando o segundo maior domínio, o paradigma "imperativo" continua dominando o mercado com 65% das indicações, seguido pelo paradigma "objeto" com 21% e o paradigma "funcional", que surge com 1% das indicações. As posições de preferência são mantidas até a quarta preferência para os paradigmas "imperativo" e "objeto", sendo que o paradigma "lógico" surge com 1% e 4% na terceira e quarta posição de domínio.

A Figura 19 é baseada nos mesmos dados, mas nos mostra uma ligeira variação em relação à Figura 18, a qual é centrada no paradigma. A Figura 19 é centrada na ordem cronológica em que os paradigmas foram aprendidos pelos profissionais, permitindo uma melhor visualização sobre a seqüência de aprendizado dos paradigmas por parte dos profissionais.

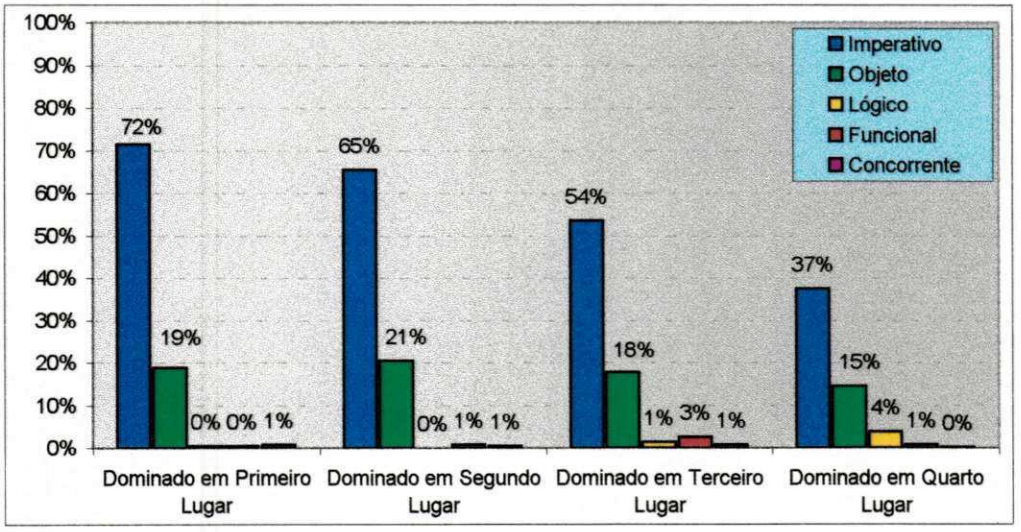

Figura 19 – Os paradigmas mais dominados pelos profissionais

Considerando a média aritmética[2] dos percentuais dos paradigmas dominados, tivemos a seguinte ordem: imperativo (57%) , objeto (18,3%), lógico e funcional (1,3% cada um) e concorrente (0,8%). O complemento percentual para 100% é atribuído às outras linguagens não contabilizadas ou ausência de resposta (vide Tabela 6).

[2] Uma forma, provavelmente mais justa, de calcular a média dos paradigmas dominados seria atribuir pesos numa escala decrescente para o paradigma dominado em primeiro lugar, em segundo lugar e assim por diante. O problema é que poderíamos ser injustos na quantificação dos pesos, ou seja, como saber quão maior deve ser o peso do primeiro lugar em relação aos demais ?

3.4.3. Sobre a dificuldade de atualização na área de programação

Na quarta parte da pesquisa fizemos a seguinte pergunta aos respondentes: "*Desde quando você se formou, que grau de dificuldade teve para se manter atualizado quanto às evoluções das técnicas de programação?*". As respostas possíveis eram: *NENHUMA, POUCA, RAZOÁVEL* ou *GRANDE*. Observamos que quase metade dos respondentes (44%) afirmam ter grande ou razoável dificuldade para se manter atualizado quanto às evoluções das técnicas de programação, enquanto que 54% afirmam ter pouca ou nenhuma dificuldade (2% não responderam).

Grande	8%
Razoável	36%
Pouca	44%
Nenhuma	10%
Não respondeu	2%

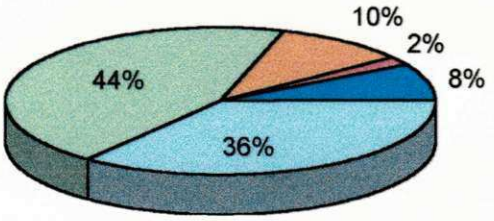

Figura 20 – Opinião dos profissionais sobre o grau de dificuldade para se manter atualizado quanto às técnicas de programação

Podemos observar que 44% de profissionais que afirmaram ter dificuldade (considerando grande ou razoável). Dentre estes, levantamos que 64% são graduados em cursos da área de informática. Assim sendo, podemos considerar que 28,16% (64% sobre 44%) dos profissionais com formação na área de informática afirmaram ter dificuldade de atualização. Da mesma forma, podemos afirmar que 15,84% (36% sobre 44%) dos profissionais que atuam na área de informática sem formação na área afirmaram ter dificuldade de atualização.

Dessa observação, podemos traçar duas análises. A primeira é que uma parcela de 28,16% de profissionais oriundos da área de informática com dificuldade de atualização pode significar alguma deficiência em sua formação, na área em análise. Afinal, o desejado seria que algo próximo de 100% dos profissionais indicassem ter pouca ou nenhuma dificuldade, demonstrando que após a quebra do vínculo com a instituição formadora, foi possível se manter aprendendo e se adaptando.

A segunda análise é sobre o fato de constatarmos que a parcela de profissionais com formação na área de informática com dificuldade de atualização é maior do que a parcela de profissionais sem formação na área. O que houve então? Para saber a resposta, algumas variáveis adicionais sobre os respondentes deveriam ser analisadas, tais como: a) se a programação é exercida como atividade fim ou como atividade meio; b) qual a intensidade e o nível de dedicação à atividade de programação dentro do trabalho do profissional; e c) a complexidade envolvida na atividade exercida (programação de sistemas comerciais,

um profissional que domina muito bem alguma linguagem do paradigma imperativo. Se a linguagem que esse profissional irá aprender for outra linguagem imperativa, o grau de dificuldade certamente será menor em relação a aprender uma linguagem de outro paradigma, tal como objeto, funcional, concorrente ou lógico. Portanto, deveremos ter um pouco de cautela na interpretação da informação aqui obtida.

Em geral, ao compararmos o grau de dificuldade de aprender uma nova linguagem declarado pelos respondentes com o tempo em que estão graduados observamos que a média do tempo decorrido desde a graduação é de 6,5 anos entre o grupo que declarou ter pouca ou nenhuma dificuldade e de 12,2 anos entre os que declararam ter grande ou razoável dificuldade. Como vemos, em média o grau de dificuldade de atualização acentua-se à medida em que cresce o tempo decorrido desde a graduação do profissional. A Figura 24 apresenta a relação entre o grau de dificuldade de aprender uma nova linguagem, o tempo decorrido desde a graduação e a experiência declarada, para cada grupo.

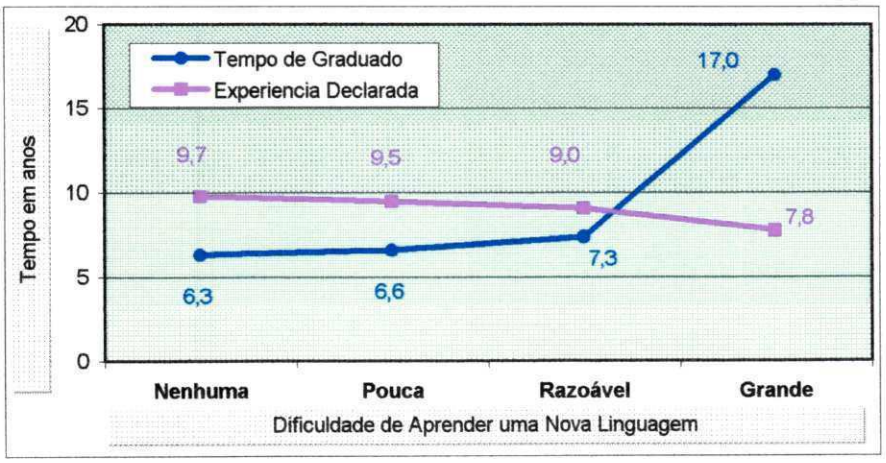

Figura 24 – Relação entre o grau de dificuldade de aprender uma nova linguagem, o tempo decorrido desde a graduação e a experiência declarada

Em seguida, solicitávamos que os respondentes procurassem indicar uma justificativa para o grau de dificuldade declarado, respondendo à pergunta "*Com que grau você acha que cada uma das opções abaixo contribuiu para este fato?*". A partir daí, dependendo do grau de dificuldade declarado, o participante iria opinar sobre um outro conjunto de possíveis motivos, sendo que para cada motivo proposto, a opinião do profissional era expressada usando a seguinte escala: "DE FORMA ALGUMA", "PROVAVELMENTE NÃO", "TALVEZ", "PROVAVELMENTE SIM" ou "CERTAMENTE".

Nos casos em que o profissional acreditasse haver algum outro motivo para o seu grau de dificuldade declarado, o qual não constasse no conjunto de motivos fornecido, ele poderia

informar livremente nos espaços reservados para "Outro", usando a mesma escala de respostas das opções pré-definidas.

Como nossa intenção era conhecer com que grau cada motivo contribuiu para o grau de dificuldade declarado, agrupamos as respostas em duas classes. A classe ***NÃO*** agrupou as respostas "DE FORMA ALGUMA" e "PROVAVELMENTE NÃO", enquanto a classe ***SIM*** agrupou as respostas "PROVAVELMENTE SIM" e "CERTAMENTE".

Para obter um valor único para cada afirmativa (o fator de influência), usamos um cálculo matemático onde as respostas da classe ***NÃO*** foram considerados como valores negativos, respostas da classe ***SIM*** como valores positivos e respostas talvez foram consideradas neutras (valor zero). Os totais de respostas das duas classes foram convertidos para percentuais e depois somados para que se eliminassem entre si, resultando na diferença entre eles. O resultado foi um valor percentual no intervalo (-100%, +100%), que mede o Fator de Influência de cada afirmativa colocada, para o grau de dificuldade declarado. A fórmula matemática usada para implementar o Fator de Influência de cada motivo foi a seguinte:

$FI_i = 100\frac{(S_i - N_i)}{(S_i + N_i + T_i)}$, onde:	**FC_i** = Fator de Influência da i-ésima afirmativa para o grau de dificuldade declarado **S_i** = número de respostas "SIM" para a i-ésimo motivo **N_i** = número de respostas "NÃO" para a i-ésimo motivo **T_i** = número de respostas "TALVEZ" para a i-ésimo motivo

Equação 2 – Cálculo do Fator de Influência dos motivos para o grau de dificuldade

A interpretação para o Fator de Influência do motivo proposto é a seguinte: um valor próximo do extremo negativo (-100%) indica uma unanimidade na afirmação de que o motivo *não* contribuiu, de forma alguma, para o grau de dificuldade. Por outro lado, um valor próximo de +100%, indica a unanimidade na afirmação de que o motivo apresentado contribuiu, com elevado grau de certeza, para o grau de dificuldade declarado. No mesmo raciocínio, valores próximos de zero indicam uma divisão de opiniões entre os participantes do grupo, ou seja, as classes de resposta ***SIM*** e ***NÃO*** ficaram equilibradas. Em resumo, qualquer que seja o resultado, ele sempre será a diferença entre as opiniões ***SIM*** e ***NÃO***.

Analisando inicialmente o grupo com maior dificuldade de aprender uma nova linguagem, mostramos, na Tabela 8, os motivos propostos e os valores obtidos para o Fator de Influência. Na tabela, os itens de **A** até **F** foram sugeridos previamente no formulário, enquanto que os itens de **G** até **L** foram obtidos depois de uma análise dos motivos espontaneamente fornecidos nos campos "Outro". A Figura 25 mostra uma representação gráfica dos Fatores de Influência apresentados na tabela, para os itens de **A** até **F**.

#	MOTIVO	SIM	NÃO	TALVEZ	FATOR DE INFLUÊNCIA
A	A forma como aprendi a programar, muito vinculada a conceitos específicos da linguagem	40%	34%	26%	6%
B	A forma como aprendi a programar, muito vinculada a conceitos da época	49%	22%	29%	27%
C	O dimensionamento inadequado da prática da programação nas disciplinas do curso	29%	47%	24%	-18%
D	A inadequação da carga horária das disciplinas de programação	31%	55%	14%	-24%
E	A pouca dedicação individual às disciplinas de programação durante a graduação	14%	72%	14%	-58%
F	A superficialidade dos conteúdos das disciplinas de programação durante a graduação	39%	41%	20%	-2%
G	Orientação insuficiente do professor ou direcionamento excessivo para a linguagem durante o curso de graduação	8%	0%	0%	8%
H	Aprendizado de linguagens ultrapassadas na graduação, desatualização curricular	5%	0%	0%	5%
I	Enfoque excessivamente acadêmico na graduação, pouca orientação para o mercado	7%	0%	0%	7%
J	Vinculação conceitual ao paradigma originalmente aprendido na graduação	8%	0%	0%	8%
K	Evolução muito rápida das técnicas e linguagens	6%	0%	0%	6%
L	Tempo decorrido desde a graduação	2%	0%	0%	2%

Tabela 8 – Os motivos responsáveis pelo maior grau de dificuldade de aprender nova linguagem e seus Fatores de Influência

Podemos observar que alguns dos motivos previamente propostos tenderam para o lado positivo, indicando que eles tiveram influência no maior grau de dificuldade dos respondentes, enquanto outros motivos tenderam para o lado negativo, revelando a possibilidade de não ter relação com o grau de dificuldade declarado.

Um Fator de Influência de 6% para o item **A** indica uma leve tendência de que a vinculação excessiva do ensino de programação com a linguagem de programação usada contribui para uma maior dificuldade de aprendizado de novas linguagens de programação ou outras inovações. A frase "muito vinculada a conceitos específicos da linguagem" se refere à possibilidade de não haver uma *separação clara* por parte do professor quanto à solução de programação adotada para um determinado problema (ex. abordagem, algoritmo) e as peculiaridades relacionadas à ferramenta usada na implementação da solução (ex. paradigma, sintaxe e semântica específica da linguagem).

A vinculação do ensino de programação com os conceitos da época (item **B**), foi considerada por quase metade do grupo (49%) como o fator que mais contribui para a dificuldade em questão.

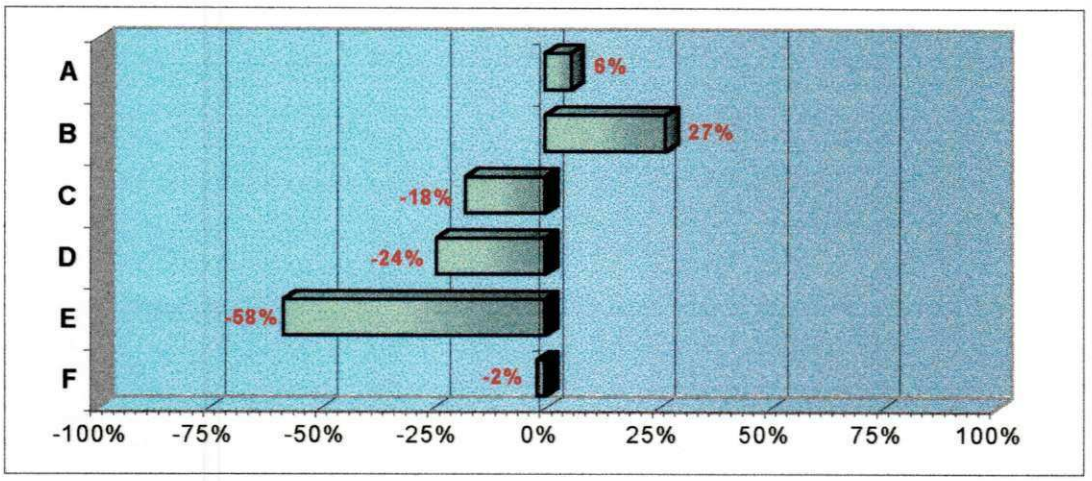

Figura 25 – Representação gráfica dos Fatores de Influência dos motivos para o maior grau de dificuldade de aprender nova linguagem

O dimensionamento inadequado da prática da programação (item **C**) não foi considerado um motivo relevante para o grau de dificuldade por 47% do grupo, enquanto que para 29% este motivo foi considerado relevante. Este resultado indica que, em geral, o grupo de maior dificuldade acredita que o dimensionamento da parte prática pode estar adequado ou não tem grande relevância para a dificuldade posterior do aprendizado de novas linguagens de programação ou outras inovações na área. O mesmo pode-se dizer do dimensionamento da carga horária das disciplinas (item **D**).

Os resultados do item **E** revelam que a despeito de gozarem de maior dificuldade, a grande maioria do grupo (72%) discorda que a pouca dedicação individual às disciplinas de programação durante a graduação seja o motivo para a dificuldade no aprendizado de novas linguagens de programação ou outras inovações na área. Como vemos, os profissionais tendem a atribuir os motivos de suas dificuldades a fatores que não dependeriam do seu próprio esforço individual. Apesar de haver a possibilidade de que esse número realmente reflita a realidade, é possível levantar-se uma questão: "qual a confiabilidade desse resultado?". Para responder, podemos citar uma pesquisa que mostra que, quando as pessoas respondem questões sobre a sua participação em equipes ou sobre o papel delas mesmas em atividades nas quais outros fatores de contribuição também estão envolvidos, elas tendem a superestimar sua própria contribuição. Por exemplo, quando perguntado individualmente aos membros de uma equipe sobre o percentual de sua contribuição na discussão sobre um projeto, a soma das opiniões geralmente ultrapassa 100%. Isto acontece porque é mais fácil lembrar da sua própria contribuição do que da contribuição dos outros membros, daí a tendência a superestimar sua contribuição. Este fenômeno foi denominado por Webb Stacy *et al* como "ruído cognitivo" (*cognitive bias*) [Stacy-95].

Ao questionarmos se o alto grau de dificuldade está relacionado com a superficialidade dos conteúdos das disciplinas de programação durante a graduação (item **F**), observamos uma divisão equilibrada de opiniões entre sim (39%) e não (41%), sendo que 20% ficou indeciso. Mesmo com uma leve tendência negativa (-2%) não é seguro concluir que a atual dificuldade de aprendizado de nova linguagem esteja ou não relacionada com o fato das disciplinas de programação terem sido muito superficiais durante a graduação.

Como podemos observar na Tabela 8, além dos motivos previamente indicados, alguns outros motivos foram sugeridos espontaneamente pelos respondentes pertencentes a este grupo. No total, 36% deles ofereceram um motivo alternativo, os quais foram agrupados e contabilizados e mostrados como itens de **G** a **L**. O valor percentual apresentado para cada motivo indica a quantidade de respondentes que forneceram espontaneamente o motivo em questão, em relação ao total de respondentes do grupo de dificuldade analisado. Todos os motivos espontaneamente fornecidos foram quantificados pelos respondentes como "CERTAMENTE", o que justifica o fato de aparecerem exclusivamente na coluna "SIM" (vide Tabela 8).

Para 8% dos membros do grupo analisado, sua maior dificuldade está vinculada ao fato de que os professores não forneceram orientação suficiente quanto à distinção sutil entre a solução de programação adotada para um determinado problema e as peculiaridades relacionadas à ferramenta usada na implementação da solução ou direcionaram excessivamente as disciplinas de programação para o ensino de linguagens de programação durante o curso de graduação (veja item **G**). Na verdade, este motivo fornecido espontaneamente é muito próximo do motivo **A**, proposto no formulário, com a diferença de que foi colocado pelos respondentes com palavras diferentes em seus comentários. De certa forma, o item **G** reforça diretamente a hipótese testada pelo item **A**. Para 5% dos respondentes do grupo, sua dificuldade está relacionada com o aprendizado de linguagens já ultrapassadas à época da graduação ou à defasagem curricular das disciplinas de programação em relação às práticas de programação da época (item **H**).

Também foi observado que 7% dos respondentes do grupo apontam como motivo da dificuldade o maior enfoque, por parte do curso de graduação (ou das disciplinas de programação) em aspectos acadêmicos de pouca aplicabilidade prática no mercado de trabalho (item **I**). Essa afirmação pode estar relacionada, por exemplo, ao fato de que os currículos adotam para o ensino de programação linguagens que não são, necessariamente, adotadas em larga escala pelo mercado (vide Tabela 1 na pág. 24 e Tabela 6 na pág. 46).

Para outros 8%, sua dificuldade se relaciona com o fato de não conseguir se desvincular conceitualmente do paradigma de programação inicialmente aprendido ao empreender

esforços para o aprendizado de linguagens de outro paradigma (item J). Embora este motivo esteja mais relacionado a um efeito do que à uma causa, ele reforça as hipóteses colocadas pelos itens **A** e **B**.

Para 6% do grupo de maior dificuldade a rápida evolução das técnicas de programação e das linguagens contribuiu para a dificuldade de aprendizado de novas linguagens ou de novos conceitos ligados à programação enquanto que 2% apontam o tempo decorrido desde a graduação (veja itens **K** e **L**). Este fato sugere que os respondentes que apontaram tais motivos não estão conseguindo se atualizar de forma eficaz com o passar do tempo, após sua formação.

Analisando agora o grupo com menor dificuldade de aprender uma nova linguagem, mostramos na Tabela 9 os motivos propostos e os valores obtidos para o Fator de Influência dos motivos no menor grau de dificuldade. Na tabela, os itens de **A** até **G** foram sugeridos previamente no formulário, enquanto que os itens de **H** até **M** foram obtidos depois de uma análise dos motivos espontaneamente fornecidos nos campos "Outro". A Figura 26 mostra uma representação gráfica dos Fatores de Influência mostrados na tabela, para os itens de **A** até **G**.

#	MOTIVO	SIM	NÃO	TALVEZ	FATOR DE INFLUÊNCIA
A	A forma como aprendi a programar, menos dependente de conceitos específicos da linguagem	67%	15%	18%	52%
B	A forma como aprendi a programar, muito independente de conceitos da época	44%	25%	31%	19%
C	O dimensionamento adequado da prática da programação nas disciplinas	31%	40%	29%	-9%
D	A adequação da carga horária das disciplinas de programação	18%	58%	24%	-40%
E	A minha experiência profissional alcançada na área de programação	73%	12%	15%	61%
F	A capacidade individual de adaptar-me a novos conceitos	79%	2%	19%	77%
G	Adequação dos conteúdos das disciplinas de programação durante a graduação	26%	41%	33%	-15%
H	Esforço individual em me atualizar e buscar conhecimento sobre tendências	5%	0%	0%	5%
I	Experiência anterior ou conhecimento preexistente na graduação	5%	0%	0%	5%
J	Aprender faz parte do meu trabalho ou é uma exigência da minha área de atuação	4%	0%	0%	4%
K	Conhecimento teórico alcançado, entendimento da diferença entre a técnica e a ferramenta (linguagem)	2%	0%	0%	2%
L	Uso simultâneo de linguagens e/ou paradigmas diferentes no meu trabalho	2%	0%	0%	2%
M	Boa qualidade do professor ou do curso na graduação	1%	0%	0%	1%

Tabela 9 – Os motivos responsáveis pelo baixo grau de dificuldade de aprender nova linguagem e seus Fatores de Influência

Dentro do grupo de menor dificuldade, 67% acredita que a maneira como aprendeu a programar influenciou em sua habilidade de aprender novas linguagens ou inovações na área de programação. A busca pela dissociação entre os conceitos de programação e as

especificidades da linguagem de programação ou uma menor dependência da linguagem, segundo a maioria deles, foi a abordagem que os conduziu para uma menor dificuldade, como indica o texto do motivo **A**. Verificamos, portanto, que há indícios de que o ensino de programação sem excessivo enfoque em linguagens de programação, pode ajudar na facilidade que o futuro profissional terá para se atualizar na área.

Foi observado também que para 44%, a desvinculação do ensino de programação com os conceitos vigentes à época de sua graduação (motivo **B**) foi um fator importante para uma menor dificuldade atual de aprender novas linguagens ou conceitos de programação, enquanto que para 25% esse motivo não foi importante. O restante do grupo (31%) ficou indeciso.

Ainda sobre o motivo **B**, podemos deixar mais claro o que significa "independência aos conceitos da época". Bem, alguns fatos ou conceitos específicos da época marcaram a evolução das linguagens e técnicas de programação. Uma delas foi a tendência em direção a níveis de abstração cada vez maiores (mnemônicos e *labels* ao invés dos códigos de operação e endereços de máquina; estruturas de controle ao invés de *jumps*; *procedures*, funções e encapsulamento ao invés de subrotinas e assim por diante). Outra tendência foi o surgimento de novos paradigmas e a crescente aceitação deles em relação ao mais antigo e predominante deles, o paradigma imperativo.

A prática mostra que, quando o ensino de programação se baseia muito nos conceitos vigentes, é possível presumir que aquele conceito enfocado evoluirá ou dará lugar a outro, ficando o profissional em dificuldade para se atualizar, devido a sua vinculação com os velhos conceitos. Se, ao invés disso, o ensino de programação abordar as tendências, dar maior enfoque na solução de problemas e preparar o futuro profissional para conviver com mudanças, pode-se supor que ele terá mais facilidade de se atualizar. Portanto, os resultados do item **B** tendem a confirmar esta hipótese.

Ao perguntarmos ao grupo se o dimensionamento adequado da parte prática da programação nas disciplinas de programação contribuiu para sua facilidade atual de aprender novas linguagens (motivo **C**), 40% respondeu que não, 31% respondeu que sim e 29% ficou indeciso. Isso sugere que, mesmo tendo alcançado uma facilidade de aprender, o dimensionamento da parte prática ficou a desejar para 40% do grupo ou caso o dimensionamento tenha sido adequado, este fato não foi decisivo.

Para apenas 18% do grupo, a adequação da carga horária das disciplinas de programação foi um fator importante para o baixo grau de dificuldade, enquanto que para 58% esse fator foi irrelevante. 24% dos respondentes do grupo não souberam ficaram indecisos (veja

motivo **D**). Esse resultado indica que, na opinião da maioria dos respondentes, a carga horária das disciplinas não estava adequada.

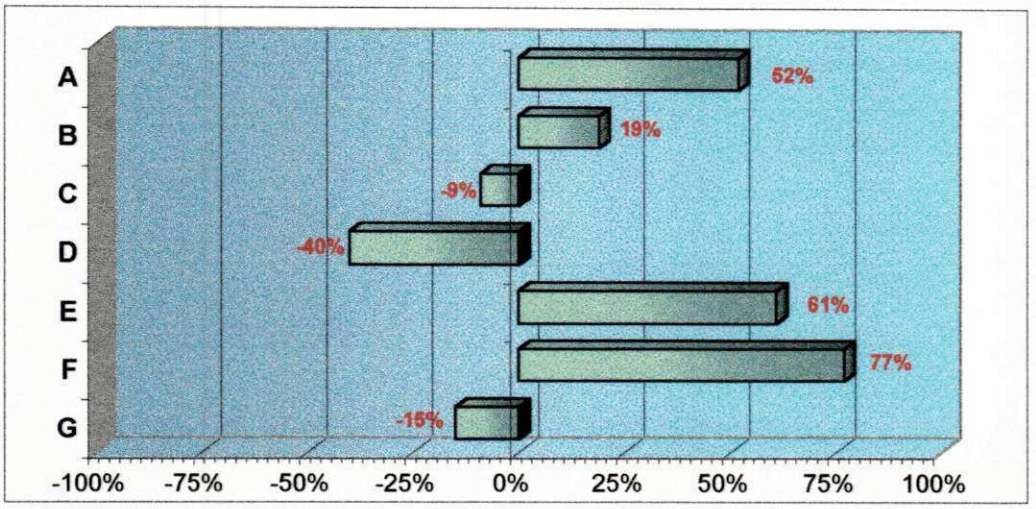

Figura 26 – Representação gráfica dos Fatores de Influência dos motivos para o menor grau de dificuldade de aprender nova linguagem

Uma grande maioria de 73% do grupo de menor dificuldade acredita que essa habilidade está relacionada com sua experiência profissional alcançada ao longo do tempo de atuação no mercado de trabalho na área de programação. Apenas 12% acredita que sua facilidade de aprender novas linguagens não tem relação com sua experiência adquirida, enquanto que 15% ficou indeciso (veja item **E**). Esse resultado indica a grande importância da atividade profissional na constante renovação de conhecimentos e manutenção do aprendizado.

Para 79% do grupo, a facilidade de aprender novas linguagens ou novos conceitos de programação se deve a sua aptidão individual de adaptação. Apenas 2% do grupo não atribui seu baixo grau de dificuldade a esse fator enquanto que 19% do grupo ficou indeciso quanto ao assunto (veja item **F**). Nesta questão, observamos o maior percentual de concordância e o menor percentual de discordância dentre todos os motivos apresentados. Neste ponto, novamente levantamos a questão do "ruído cognitivo" que afeta os profissionais quando estes opinam sobre si próprios, podendo levá-los a superestimar suas próprias qualidades (veja discussão do item E da Tabela 8).

Quanto ao item **G**, 26% do grupo concorda que a adequação dos conteúdos das disciplinas de programação na sua graduação foi um fator que contribuiu para o baixo grau de dificuldade, enquanto que 41% do grupo discorda e 33% ficou indeciso. Aqui tivemos o maior percentual de indecisão e um dos menores percentuais de concordância. A maioria, portanto, não acredita que os conteúdos eram adequados o suficiente para influenciar no baixo grau de dificuldade.

O esforço individual em buscar atualização, maiores conhecimentos e informações sobre tendências na área de programação surgiu espontaneamente como um dos motivos para baixo grau de dificuldade de 5% dos respondentes (veja item **H**). Da mesma maneira, o fato de já dispor de conhecimentos de programação ou experiência profissional antes da graduação foi importante para o atual baixo grau de dificuldade para 5% do grupo (veja item **I**).

Para 4% do grupo, sua facilidade em aprender novas linguagens ou acompanhar inovações na área de programação também tem relação com o tipo de trabalho que executa como profissional (veja item **J**). Para eles, o aprendizado de linguagens de programação é um componente das suas atribuições como profissional. Por exemplo, aqueles que exercem atividades ligadas à docência aprendem porque precisam ensinar e essa prática constante lhes legou, com o tempo, a habilidade de entender facilmente os diferentes dialetos da programação. Outros profissionais que apontaram ter facilidade de aprender novas linguagens relataram a necessidade de freqüente contato com inovações na área de programação como uma exigência do tipo de trabalho que exercem, o que lhes trouxe mais habilidade para aprender novas linguagens de programação.

O conhecimento teórico e o entendimento da diferença entre "programar" e "codificar" alcançados ao longo da graduação e da prática profissional foi citado por 2% do grupo como de grande relevância para o baixo grau de dificuldade em aprender novas linguagens ou acompanhar inovações na área de programação (veja item **K**).

A necessidade de usar simultaneamente mais de uma linguagem de programação ou utilizar linguagens de diferentes paradigmas no exercício profissional foi colocado como fator contribuinte para o baixo grau de dificuldade para 2% desse grupo (veja item **L**).

Apenas 1% do grupo creditou espontaneamente seu baixo grau de dificuldade à boa qualidade do curso de graduação ou à qualidade da atuação dos professores (veja item **M**).

Parte II

4. Discussão

Antes de iniciarmos maiores discussões quanto aos resultados obtidos pelas pesquisas apresentadas, faremos algumas considerações sobre a metodologia usada nas pesquisas e sobre as limitações dos resultados obtidos.

Basicamente o tipo das pesquisas que conduzimos apresenta dois problemas: seleção e amostragem [Marconi-90]. Quanto ao mecanismo de seleção, existem dois tipos: a *auto-seleção* e a *seleção censitária*. A auto-seleção é o tipo de seleção na qual as pessoas dispõem da faculdade de participar ou não da pesquisa, ou seja, elas podem selecionar a elas mesmas ou não. Por exemplo, quando uma pessoa desliga o telefone em uma pesquisa baseada nesse método, uma auto-seleção ocorreu, pois ela decidiu não participar após ter recebido o convite. Da mesma forma, quando um participante em potencial devolve uma carta-resposta uma auto-seleção ocorreu, pois ele decidiu participar. Ou seja, um elemento da população[5] alvo tem a opção de participar ou não da pesquisa e pode decidir por si mesma.

As pesquisas que adotam a seleção censitária, por sua vez, não oferecem a opção aos elementos da população alvo de participarem ou não: elas são obrigadas a participar. Alguns exemplos são um censo populacional ou uma eleição política para presidente da república (neste caso a população alvo é composta pelos eleitores, segundo a lei vigente).

Se uma pesquisa, tal como as nossas, se baseia na auto-seleção como mecanismo de seleção, isto significa que alguma opinião desejada ou importante poderá ter sido omitida, uma vez que as pessoas podem optar por participar ou não. Nesses casos, como não se tem a opinião de todos, a opinião de um grupo representativo é usada para inferir a opinião de um grupo maior, suscitando a possibilidade de erros e desvios. Na verdade, a grande maioria das pesquisas sofrem este tipo de problema.

Quanto à amostragem, existem essencialmente dois tipos: aleatória e não-probabilista. A seleção é aleatória (ou probabilista) quando cada membro da população tem a mesma probabilidade de ser escolhido. Nossa amostragem é do segundo tipo, a não-probabilista, a qual se baseia na população atingida pelo anúncio ou convite para participar. Como certamente o convite não foi enviado para todas as pessoas que atendem ao perfil do respondente, uma parte do universo de respondentes potenciais não tomou conhecimento da pesquisa e, portanto, não pôde participar (ex. nem todos têm acesso *à Web* ou correio eletrônico). Como conseqüência, nem todos os segmentos de opinião podem estar

representados nos resultados da pesquisa e, embora todos os participantes atendam às exigências da amostra, há uma redução potencial na possibilidade de generalizar os dados obtidos para todos os demais membros do universo que atende ao perfil exigido. Portanto, a técnica de amostragem não-probabilista é utilizada quando alguns fatores impedem a escolha da população total ou de uma amostra probabilista, ficando a cargo do pesquisador buscar uma amostragem representativa. Esta observação atinge particularmente a pesquisa 2 sobre "A prática do profissional de programação", uma vez que o universo de profissionais em todo o mundo é incerto, dificultando a comparação do tamanho da amostra com o tamanho da população total.

A característica principal das técnicas de amostragem não-probabilistas, como a nossa, é a de que, não fazendo uso de formas aleatórias de seleção, torna-se desaconselhável a aplicação de fórmulas para tratamento estatístico mais complexos, tais como cálculo de erros de amostra e desvio [Marconi-90].

Estas considerações aqui colocadas visam estabelecer as limitações impostas pelo método usado, de modo que a interpretação dos resultados seja cuidadosa e responsável, dentro de uma amplitude realista. Assim, acreditamos que as interpretações podem ser um bom começo para avaliar a opinião dos docentes sobre a prática do ensino de programação.

A despeito das limitações quanto à aplicação de tratamentos estatísticos mais refinados, esta pesquisa tenta mostrar uma ligação entre a aquisição do conhecimento nos cursos de graduação e a efetiva prática do profissional inserido no mercado de trabalho, fornecendo um *feedback* mínimo para qualquer avaliação e redirecionamento das práticas pedagógicas vigentes na área de programação.

Uma vez analisados isoladamente os resultados obtidos nas pesquisas 1 e 2, tentaremos neste capítulo fazer uma análise mais ampla, interrelacionando os resultados das pesquisas e tentando extrair delas alguns ensinamentos que possam contribuir para a proposição de melhorias no ensino de programação. Na seção 4.1, tentaremos realçar alguns aspectos relevantes que foram observados nas pesquisas, os quais podem apontar algumas ações ou estratégias para a aprendizagem continuada, tais como a relação existente entre a academia e o mercado quanto à adoção de linguagens de programação e a relevância da modernização curricular na área de computação. A seção termina com uma avaliação crítica sobre a implementação das pesquisas realizadas, o que inclui algumas orientações sobre pontos que podem ser evitados ou enfatizados em trabalhos futuros nessa direção.

---

[5] Entende-se por "população" o conjunto de pessoas que apresentam pelo menos uma característica em comum [Marconi-90]. No nosso caso, a população atende ao perfil estabelecido para o "público alvo".

Em seguida, nas seções 4.2 e 4.3, buscaremos ampliar nossos conhecimentos sobre a atividade de programação e sobre os aspectos cognitivos envolvidos no aprendizado de programação. Afinal, o que está envolvido na tarefa de programar? Quais as dificuldades relacionadas com o ensino e com o aprendizado de programação? Esta seção busca evidenciar a relação existente entre a programação e a área da psicologia cognitiva, através da compilação de alguns trabalhos importantes que abordam o tema. A discussão sobre os mecanismos cognitivos envolvidos e sobre dificuldades e conflitos vividos pelo aluno no aprendizado de programação é importante para que possamos identificar melhores abordagens em nossa prática pedagógica e direcionar esforços visando um processo ensino-aprendizagem mais eficaz.

O capítulo culmina, na seção 4.4, com a proposição de algumas estratégias, tanto no âmbito acadêmico quanto administrativo, que podem contribuir para melhorar o ensino de programação e favorecer a aprendizagem continuada dos alunos e egressos. As estratégias se baseiam na análise dos resultados das pesquisas realizadas e na discussão estabelecida sobre os aspectos cognitivos envolvidos no aprendizado e na prática da programação.

4.1. O Que aprendemos com as pesquisas

4.1.1. Linguagens de programação: a academia e o mercado

Como foi possível observar, o uso das linguagens de programação difere na academia e no mercado de trabalho. É possível entender esse fato analisando os diferente objetivos com que a programação é atacada. Na academia, a linguagem de programação é usada como um instrumento pedagógico na transferência de conhecimentos curriculares diversos, que variam desde o ensino de conceitos de computação, desenvolvimento de perfis associados a várias unidades de conhecimento até a própria transferência da competência da programação em si. A programação, portanto, tem ampla aplicabilidade, permeia todo o currículo e transcende os limites da programação em si mesma, ou seja, tanto se usa programação para ensinar quanto se ensina programação. Partindo desse ponto de vista e considerando que a linguagem deve ser uma aliada e não uma elemento de obstrução, a escolha da linguagem de programação para a academia adota prioritariamente critérios como legibilidade, simplicidade sintática e semântica, clareza de estrutura e alto nível.

O mercado de trabalho faz uso da programação profissional, geralmente voltada para nichos específicos de aplicação (grandes e pequenos sistemas comerciais, aplicações de tempo real, software básico e outros), os quais podem exigir linguagens especializadas. Programadores profissionais geralmente participam de equipes de trabalho e também convivem com soluções que envolvem a programação em larga escala (*programming at*

large) além das pequenas soluções de fronteiras reduzidas (*programming in the small*). Esta prática mais objetiva e de forte relacionamento com a engenharia de software tem objetivos nitidamente contrastantes com os da academia, os quais levam à adoção de outros critérios para a escolha da linguagem de programação, tais como a eficiência, flexibilidade, granularidade e controle, concisão, acessibilidade (aos recursos da máquina), entre outros.

De fato, as pesquisas mostraram isso. Na pesquisa 1, observamos que a academia tem uma grande preferência pela linguagem Pascal, para o ensino da primeira linguagem de programação, pelo fato de ser considerada uma linguagem com uma sintaxe simples, altamente estruturada e fortemente tipada, provendo um ambiente "seguro" para programadores novatos. A pesquisa 2, entretanto, apontou o domínio predominante da linguagem C entre os profissionais, confirmando a adoção dos critérios mencionados.

4.1.2. Modernização curricular

Os resultados obtidos pela Pesquisa 1 quanto às linguagens ensinadas se referem predominantemente ao currículo de 1996-1997. Por outro lado, a média do tempo decorrido desde a graduação para os profissionais que participaram da Pesquisa 2 é de 7 anos, o que significa que as linguagens de programação que aprenderam reportam-se ao contexto curricular típico dos cursos de graduação no final da década de 80. Isso explica, em parte, o fato de haver diferenças nos resultados obtidos pela Pesquisa 1 quanto às linguagens ensinadas e os resultados obtidos pela Pesquisa 2 quanto às linguagens aprendidas (vide Tabela 3 e Tabela 5 nas páginas 32 e 43, respectivamente). Por outro lado, este fato também realça a busca pela atualização curricular, por parte das instituições, no sentido de adotar novas linguagens e modernizar seus currículos, embora linguagens ultrapassadas, pelo menos do ponto de vista didático, a exemplo de COBOL e FORTRAN, ainda permaneçam em alguns currículos brasileiros, mesmo a pretexto de suporte a sistemas legados.

Ao compararmos os paradigmas ensinados atualmente (em 1996-1997) e os aprendidos pelos profissionais em períodos anteriores[6] (veja Tabela 10), verificamos que o paradigma imperativo continua, como antes, sendo a preferência para o ensino da primeira linguagem de programação, embora outros paradigmas tenham ocupado mais espaço no ensino da segunda e terceira linguagens de programação. O paradigma de orientação a objeto cresceu em preferência no ensino da primeira, segunda e terceira linguagens mas manteve sua vocação de ser ensinada preferencialmente como segunda e terceira linguagens. O

[6] Os dados são baseados nas informações dos respondentes da Pesquisa 2, que representam uma população de ex-estudantes que concluíram a graduação entre 1956 e 1997, com média em 1990.

paradigma lógico continua uma opção desconsiderada para o ensino da primeira linguagem de programação, embora tenha crescido na preferência para o ensino da terceira e da segunda linguagem, nessa ordem. O uso de linguagens funcionais perdeu a pouca preferência que tinha para o ensino da primeira linguagem de programação (caiu de 1% para 0%), mas cresceu como opção para o ensino da terceira e segunda linguagens, nessa ordem. O uso de linguagens do paradigma concorrente, continua, como antes, longe dos currículos dos cursos de graduação brasileiros.

PARADIGMA	1º LUGAR		2º LUGAR		3º LUGAR	
	ATUALMENTE	ANTIGAMENTE	ATUALMENTE	ANTIGAMENTE	ATUALMENTE	ANTIGAMENTE
IMPERATIVO	94%	94%	50%	82%	25%	66%
OBJETO	6%	1%	32%	5%	45%	12%
LÓGICO	0%	0%	14%	3%	15%	3%
FUNCIONAL	0%	1%	4%	2%	15%	3%
CONCORRENTE	0%	0%	0%	0%	0%	0%

Tabela 10 – A adoção dos paradigmas de programação no ensino: passado e presente

Podemos observar, portanto, que o uso de linguagens imperativas manteve-se estável como opção para o ensino da primeira linguagem de programação, mas perdeu seu espaço para os paradigmas orientação a objetos, lógico e funcional na preferência para o ensino da segunda e terceira linguagens.

Além disso, os benefícios do paradigma de orientação a objeto foram reconhecidos pela academia mas sua adoção para o ensino de primeira linguagem foi aparentemente cautelosa, a julgar pelo uso ainda tímido em relação ao que o mercado já experimenta desse paradigma. Entretanto, há indícios de que a presença desse paradigma cresça no ensino da primeira linguagem de programação nos próximos anos com a adoção de linguagens como Java e o crescimento da participação de C++ nos currículos (vide Tabela 3, pág. 32).

4.1.3. O que deixamos de aprender: uma crítica sobre a implementação das pesquisas

Uma das etapas mais importantes na elaboração de um questionário ou formulário de pesquisa é a da elaboração das hipóteses. Uma hipótese é uma proposição que se faz na tentativa de verificar a validade das respostas para um problema, uma suposição que antecede a constatação dos fatos e tem como característica uma formulação provisória [Marconi-90]. Uma vez que a função da hipótese, na pesquisa científica, é propor explicações para certos fatos e ao mesmo tempo orientar a busca de outras informações, a clareza da definição dos termos da hipótese é condição de grande importância para o desenvolvimento da pesquisa.

Muitas vezes, entretanto, nos estudos de caráter exploratório ou descritivo, tal como as pesquisas que desenvolvemos, algumas dificuldades devem ser vencidas na tarefa de elaborar as hipóteses, principalmente aquelas advindas da escassez de referências teóricas e práticas. Em nosso caso particular, o conjunto de referências teóricas existente sobre o ensino de programação é bastante rico, embora poucos tenham sido os trabalhos práticos encontrados sobre o processo ensino-aprendizagem na área de programação. O estabelecimento das hipóteses de trabalho, portanto, foi baseado principalmente em aspectos informais do ensino de programação, na experiência adquirida pela prática do ensino e no sentimento de alguns professores e em observações no campo profissional.

Na fase de análise e interpretação dos dados coletados na pesquisa, foi possível perceber alguns pontos que poderiam ter sido explorados com maior nível de detalhe, revelando que poderíamos aprender um pouco mais com as pesquisas. Em outros casos, algumas questões poderiam ter sido explicitadas com maior clareza, eliminando potenciais pontos de falha no formulário.

Por exemplo, na Pesquisa 2, sobre a prática do profissional de programação, na parte 4, perguntamos "*Desde quando se formou, que grau de dificuldade teve para se manter atualizado quanto às evoluções das técnicas de programação?*". Ficamos satisfeitos em conhecer que uma parcela de 8% dos respondentes tem grande dificuldade, 36% tem razoável dificuldade, 44% tem pouca e 10% tem nenhuma dificuldade. No entanto, não exploramos o suficiente para conhecer se a principal dificuldade é ligada a aspectos financeiros, cognitivos, disponibilidade de tempo, disponibilidade de opções ou outro motivo qualquer.

Quanto aos potenciais pontos de falha nos formulários, eles ocorrem quando a formulação das questões dá margem para diferentes interpretações. Nesses casos, não é possível ter certeza se todos os respondentes entenderam da mesma maneira o objetivo da questão e corre-se o risco de coletar respostas iguais para interpretações diferentes. Portanto, as questões devem ser claras e, ao mesmo tempo, concisas.

Entretanto, a questão da concisão X clareza não é um problema tão trivial quanto parece. Um formulário de pesquisa deve ser o menos extenso possível, sob o risco de desmotivar o respondente e vê-lo abandonando o questionário sem concluir seu preenchimento, principalmente quando o veículo é a Web. De fato, alguns estudos revelam que existe uma correlação entre o tempo médio de preenchimento de um formulário de pesquisa pela Web e o índice de desistência durante o seu preenchimento [Pitkow-94]. Por esse motivo, pesquisadores que utilizam a Web como veículo para *surveys* devem criteriosamente determinar a quantidade de questões, o tamanho do seu enunciado e a complexidade

envolvida na resposta (observando a concisão como redutor do tempo envolvido na resposta do formulário), ao mesmo tempo em que preza pela clareza com que são apresentadas.

Muitas vezes, portanto, é difícil ser claro sendo conciso. Este talvez seja o caso do item **A** da parte 6a da Pesquisa 2, onde colocamos a alternativa "A forma como aprendi a programar, muito vinculada a conceitos da linguagem" como provável motivo para grau de dificuldade declarado (vide Tabela 8, na pág. 55). Como já foi colocado no texto que interpreta os resultados para esse item, a frase "muito vinculada a conceitos específicos da linguagem" se refere à possibilidade de não haver uma separação clara por parte do professor quanto à solução de programação adotada para um determinado problema e as peculiaridades relacionadas à ferramenta usada na implementação da solução. O problema é: será que todos entenderam da mesma forma o que estava sendo implicitamente proposto? O surgimento espontâneo entre os respondentes de um motivo que é praticamente o que já propúnhamos no item **A** (vide item **G**, sintetizado a partir das respostas "Outro"), é uma evidência de que a concisão imposta ao item **A** pode ter afetado sua clareza.

4.2. Mecanismos cognitivos envolvidos no aprendizado e na prática de programação

Os campos de estudo da cognição e da programação são relacionados de várias maneiras. Primeiro, a psicologia cognitiva é baseada em um 'modelo computacional', no qual a mente humana é vista como um tipo de processador de informações, similar a um computador. Segundo, a psicologia cognitiva oferece alguns métodos para se examinar o processo que se esconde por trás da execução de algumas tarefas de computação, mais especificamente a programação. Por último, a programação é uma tarefa bem definida e existe um grande número de programadores profissionais, o que a torna uma tarefa ideal para o estudo de processos cognitivos no domínio do mundo real.

A psicologia cognitiva é o estudo dos mecanismos pelos quais os processos mentais são conduzidos, e dos tipos de conhecimentos requeridos para cada processo. Os processos cognitivos incluem percepção, atenção, armazenamento e recuperação de informação na memória humana, produção e entendimento de linguagens, solução de problemas e raciocínio [Ormerod-90].

Existem estudos que demonstram que os programadores constróem em suas mentes estruturas multi-dimensionais que são manipuladas no processo de criação, especificação, codificação e simulação do comportamento de programas [Petre-97], [Fix-93], [Stacy-95].

Embora não seja nossa intenção discutir os detalhes da psicologia cognitiva, dada a amplitude e a profundidade que o tema encerra, teceremos alguns comentários sobre os aspectos ligados ao aprendizado de programação e à prática do profissional.

Muitos estudos empíricos já foram feitos na tentativa de explicar e documentar os mecanismos cognitivos envolvidos na tarefa de programação. Embora a comunidade científica não detenha ainda o conhecimento completo sobre tal assunto, já foi possível formular algumas teorias que podem contribuir para melhorar a forma como ensinamos nossos alunos a programar e a desenvolver uma prática adequada na formulação de soluções para os problemas que lhes são apresentados.

De acordo com o que foi observado em comentários dos participantes da Pesquisa 1, muitas abordagens usadas em cursos de introdução à programação enfocam tipicamente a sintaxe e a semântica das linguagens. Novos estudos com programadores novatos, entretanto, sugerem que o maior problema não é a linguagem para a construção da solução, mas a construção da solução em si. Ou seja, a maior dificuldade é "juntar as peças", conceituar, coordenar e compor os componentes do programa [Soloway-86], [Martin-96].

Para entendermos melhor como transferir para os alunos os conhecimentos e habilidades envolvidas na programação, conceituaremos a seguir o que é "programação" e teceremos alguns comentários sobre as dificuldades inerentes ao aprendizado da programação, a forma como novatos e experientes encaram a programação, alguns aspectos cognitivos da atividade profissional de programação e, finalmente, apresentaremos uma compilação de critérios que devem ser usados na adoção das linguagens para o ensino de programação.

4.2.1. O que é programação

Segundo Allen Tucker [Tucker-91], o termo *programação* é entendido para denotar toda a coleção de atividades que cercam a descrição, desenvolvimento e implementação efetiva de soluções algoritmicas para problemas bem definidos. Para Brooks (Brooks apud [Pennington-90]:46), programação consiste em uma série de mapeamentos de um domínio de conhecimento para outro, passando por vários domínios de conhecimento intermediários e terminando com o domínio de conhecimento da linguagem de programação.

A programação de computadores foi caracterizada por Greeno e Simon (Greeno e Simon apud [Pennington-90]:46), dentro de uma conceituação mais ampla, como uma tarefa relacionada a projeto, tal como arquitetura, coreografia, projeto de um circuito elétrico, projeto de uma obra civil e outros. Segundo eles, a programação envolve juntar um conjunto de instruções, em uma determinada linguagem, as quais irão resolver um problema definido.

Em geral, a tarefa de projetar envolve duas atividades de transformação fundamentais, que são a *composição* e a *compreensão*. A essência da composição na programação é o mapeamento do *quê* o programa irá realizar, sob o ponto de vista do domínio do problema no mundo real, para uma detalhada lista de instruções para o computador, indicando *como* alcançar aqueles objetivos, no domínio da linguagem de programação. Já a compreensão de um programa pode ser vista como o caminho inverso, ou seja, uma série de transformações partindo do *como* para o *quê* [Brooks-77].

Ambas as transformações de composição e compreensão são atividades psicologicamente complexas, por envolverem sub-tarefas que se desenvolvem em diferentes domínios de conhecimento e carreiam uma variedade de processos cognitivos. Por exemplo, a programação pode ser dividida nas seguintes sub-tarefas: entendimento do problema, planejamento da solução (método de abordagem) e codificação. Adicionalmente, poderíamos citar outras sub-tarefas, tais como a manutenção e documentação.

A Figura 27 caracteriza de forma geral as sub-tarefas da programação, o seqüenciamento em que se dão os processos básicos de composição ou compreensão, o domínio de conhecimento necessário a cada tarefa, uma descrição do modelo que se forma durante o processo cognitivo (representação mental) e a representação externa relacionada àquela sub-tarefa.

Sub-tarefas da Programação	Processo Básico		Domínio de Conhecimento	Representação Mental	Representação Externa
Entendimento do problema	COMPOSIÇÃO ↓	COMPREENSÃO ↑	Domínio específico do problema (finanças, física, geometria, estatística, comércio, indústria etc)	Modelo de situação	Documentação de requisitos e de especificação
Projeto, planejamento			Estratégias de projeto + Algoritmos + Métodos de programação	Modelo de solução Representação do plano	Documentação do projeto
Codificação			Linguagens de programação + Convenções sintáticas e semânticas + Estilos, paradigmas	Representação do programa	Código
Manutenção			Todos os anteriores + Depuração, estratégias de testes, tipos freqüentes de erro	Todas as representações	Todos os documentos

Figura 27 – Caracterização das tarefas relacionadas com a programação e os processos envolvidos

A separação clara entre cada sub-tarefa, didaticamente mostrada na figura, contrasta com o que usualmente ocorre em projetos de software mais complexos. Na prática, as sub-tarefas apresentam múltiplas interconexões, o que as tornam difíceis de separar, uma vez que a conceitualização, o projeto (planejamento) e a implementação irão influenciar uns aos outros em todas as direções. Um exemplo disso é o fato de que os programadores raramente completam uma sub-tarefa antes de passar a outra. Ao invés disso, o programador estabelece um processo de sucessivas alternações entre as sub-tarefas, enquanto continua trabalhando no "entendimento do problema". Ou seja, o programador

trabalha no problema enquanto projeta, codifica e revisa, caracterizando um ciclo de refinamento sucessivo [Pennington-90].

Sob a perspectiva das atividades cognitivas envolvidas, Rogalski e Samurçay [Rogalski-90] constataram que as dimensões cruciais ligadas à tarefa de programação são o *planejamento*[7] e a *representação*. A Figura 28 apresenta um esquema que permite analisar as atividades cognitivas envolvidas nas tarefas da programação, partindo de um problema do mundo real até chegar em um texto de programa pronto para executar em um determinado dispositivo computacional.

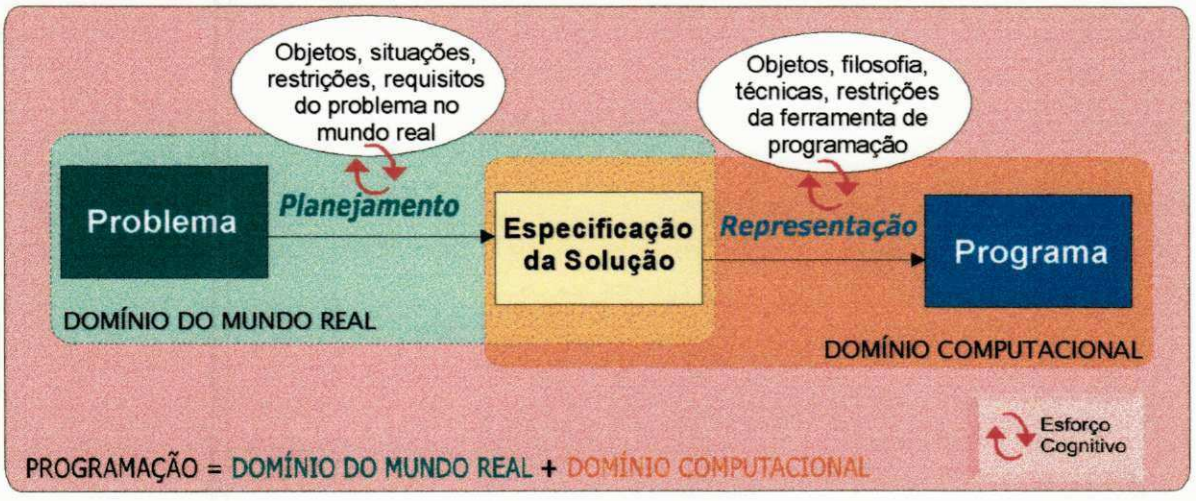

Figura 28 – A atividade de programação: do problema ao programa

Como vemos na figura, a programação é uma composição das duas tarefas *planejamento* e *representação*, as quais têm a *especificação da solução* como elemento comum. Ou seja, a especificação da solução é o elemento de chegada (alvo) da tarefa de planejamento e o elemento de partida (origem) da tarefa de representação.

Na programação, portanto, o problema do mundo real é primeiro resolvido no domínio onde ocorre (domínio do mundo real), gerando-se uma especificação da solução (que pode ser um documento formal ou, em geral, apenas um modelo mental), compondo um esforço cognitivo chamado de *planejamento*. Como visto na figura, na tarefa de planejamento os objetos e suas situações específicas, os requisitos funcionais e as restrições particulares do problema do mundo real são os elementos ativos que interagem com esforço cognitivo do programador (resolvedor do problema) para que seja alcançada uma solução.

[7] Os autores denominam os processos envolvidos de *processing* e *representation*, os quais são explicados no contexto na psicologia cognitiva. Para evitar possíveis confusões com o termo "processamento", o qual tem forte significado no domínio computacional, adotaremos o termo "planejamento" em substituição à tradução literal do termo *processing*, por julgarmos ser mais adequado em nossa língua.

A partir da especificação alcançada para a solução do problema (domínio do mundo real), o programador empreende um esforço cognitivo adicional para representar a solução no domínio da ferramenta de programação (domínio computacional), compondo a etapa chamada de *representação*. Como visto na figura, a representação consiste na transformação de uma especificação da solução em um programa, isto é, em um texto que pode ser interpretado sem dubialidades por uma máquina para calcular o resultado da função. Desse ponto de vista, considera-se que o esforço cognitivo envolvido está relacionado à instanciação da solução, ou seja, ao mapeamento da solução para o programa, usando uma linguagem de programação. Em resumo, na representação, a atividade do programador passa a ser a tarefa de adequar a solução a um conjunto de regras, restrições e filosofia (paradigma), específicos da linguagem, usando conceitos e técnicas de programação conhecidas, produzindo a instanciação da solução: o programa.

Novamente ressalta-se o caráter didático do esquema descrito, uma vez que, na prática, não é trivial separar com clareza cada etapa ou processo cognitivo durante a tarefa de solucionar um problema. Entretanto, o *planejamento* deve normalmente preceder a *representação*, embora seja aceitável alguma interveniência do domínio computacional sobre o planejamento da solução, em virtude de possíveis restrições de implementação, caracterizando-se um processo de realimentação e refinamento. O *planejamento* como atividade cognitiva é vital para manter uma imagem mental da solução durante o processo de *representação*.

Quando programadores estabelecem a prática de abordar o problema partindo tipicamente do esforço cognitivo de *planejamento* (domínio do mundo real) para, em seguida, buscar a *representação* (domínio computacional), tendem a desenvolver a habilidade de armazenar esquemas mentais para classes de problemas comuns, que o ajudarão em problemas subseqüentes, acumulando-se, assim, experiência na solução de problemas.

Quando a *representação* é acionada antes do *planejamento* ou quando este é minimizado ou ignorado, o enfoque da solução do problema recai sobre a codificação, podendo resultar em soluções excessivamente dirigidas à ferramenta. Na prática, isso ocorre quando o problema é abordado primeiramente no domínio computacional, ou seja, quando o programador aborda o problema partindo diretamente para a codificação ou quando a solução é "pensada" em termos da linguagem de programação.

Os estudos sobre aquisição de habilidades de programação são principalmente centrados na dimensão do *planejamento*. Uma das razões para isso é a importância do entendimento e abordagem do problema na programação, onde a organização antecipada das ações tem papel de grande influência na produtividade da tarefa de programação. Além disso, há um

reconhecimento de que a construção da solução em si é mais importante, do ponto de vista do processo cognitivo, do que a representação dela em uma linguagem, ou seja, sua codificação [Rogalski-90].

Em resumo, programação é uma atividade cognitiva complexa, composta por uma variedade de sub-tarefas que se interrelacionam e envolvem vários tipos de conhecimento especializado. A aquisição de conhecimento e habilidades sobre programação também é um processo complexo, difícil de se estudar e com muitas questões ainda em aberto, embora seja melhor dominado nos dias de hoje graças aos trabalhos multidisciplinares desenvolvidos no campo da psicologia da programação.

4.2.2. Aspectos individuais e relacionados ao trabalho em grupo

Como em outras áreas, muito do trabalho de um profissional de computação se dá em equipes. Uma equipe é um grupo de pessoas organizado para trabalhar junto e alcançar um conjunto de metas que não poderiam ser alcançadas individualmente com a mesma qualidade e eficácia.

Entretanto, os benefícios da qualidade e eficácia superiores advindos do trabalho colaborativo passa pela superação de algumas barreiras que não estão presentes no trabalho individual, uma vez que os mecanismos cognitivos envolvidos na solução de problemas e definição de requisitos enfrentam significativas limitações quando o processo se dá em uma equipe, ao invés de individualmente. Isso se dá pela dificuldade do entendimento coletivo do problema e da formulação cooperativa da sua solução.

A diversidade do entendimento da natureza e dos limites do problema, as intenções individuais dos atores envolvidos, expectativas, objetivos, conhecimento, experiência, disponibilidade de recursos, etc, podem facilmente criar conceitualizações alternativas da "solução" a desenvolver. Este problema que se dá nas equipes foi abordado por Butterfield *et al* e denominado de "conflito cognitivo" [Butterfield-94]. Logo, a eficiência do processo de definição dos requisitos é uma função da habilidade do grupo extrair o pertinente conhecimento e representação do problema de cada indivíduo e organizar uma solução mutuamente aceitável.

Como podemos notar, o trabalho em equipe envolve o problema extra de remediar as diferentes interpretações e entendimentos que são inerentes ao grupo, o que é alcançado por um processo de intensa e efetiva interação, cooperação e comunicação.

4.2.3. Criação X manutenção de programas

Muitas vezes, a tarefa do programador pode ser diferente de solucionar um problema do mundo real e, em seguida, criar um programa (mecanismo de composição). Em certos casos, a tarefa se relaciona com a análise de um programa já escrito (mecanismo de compreensão), seja para adaptá-lo, melhorá-lo, corrigi-lo ou reutilizá-lo. Portanto, a manutenção também é uma atividade importante na vida profissional do programador, dada a freqüência com que ocorre em sua prática como profissional.

O problema da criação de um programa é o estabelecimento de um mapeamento entre as entidades do mundo real e suas representações no programa. Na compreensão de programas, o problema é fazer o mapeamento reverso a partir de um dado programa para o entendimento das entidades do mundo real e suas ações envolvidas [Ramalingam-97].

Algumas teorias foram formuladas e estudos foram conduzidos para explicar o comportamento relacionado à solução de problemas nos programadores envolvidos em manutenção e como eles buscam o entendimento dos programas. Os diferentes modelos cognitivos são usualmente variações dos modelos "entendimento *top-down*", "entendimento *bottom-up*", "refinamento interativo de hipóteses" ou alguma combinação destes [Good-97b], [Letovsky-86].

A abordagem "*bottom-up*" reconstrói o projeto de alto nível do sistema, começando pelo código fonte através de uma série de passos que envolvem a separação de trechos e uma atribuição de conceitos. A abordagem "*top-down*" começa com uma noção pré-existente da funcionalidade do sistema e prossegue indicando quais segmentos ou componentes do sistema são responsáveis por tarefas específicas. O método de refinamento interativo envolve a criação, verificação e modificação de hipóteses até que todo o sistema seja explicado usando-se um conjunto de hipóteses consistentes.

A experiência mostra que mantenedores de sistemas regularmente trocam entre estes diferentes modelos dependendo do problema ou tarefa em suas mãos. Consequentemente, nenhum modelo trivial explica tudo do comportamento cognitivo envolvido na compreensão de programas, uma vez que um modelo interage com outro em vários níveis e em diferentes momentos.

4.3. Dificuldades no aprendizado de programação

Como vimos na seção 4.2, a atividade de *programação* é mais ampla do que a tarefa de simplesmente codificar em uma linguagem de programação. Os mecanismos cognitivos envolvidos são distribuídos ao longo de duas atividades conexas, que são o *planejamento* e

a *representação*, que se dão no domínio do mundo real e no domínio computacional, respectivamente.

A atividade de "ensinar programação", por outro lado, requer do professor o conhecimento sobre as principais dificuldades enfrentadas pelos alunos para entender os conceitos da programação e a sensibilidade para perceber certas diferenças na maneira de ensinar para novatos e para experientes. Classificamos o programador como "novato" quando ele está aprendendo a programar pela primeira vez e não domina suficientemente uma linguagem de programação para construir um programa que represente a solução de um problema. Tipicamente o programador novato é um aluno da disciplina introdutória de programação (Programação I ou Linguagens de Programação I, dependendo do currículo).

Consideramos o programador como "experiente" quando ele é não-novato, ou seja, quando demonstra conhecimentos sobre resolução de problemas, técnicas de programação, estrutura de dados e domínio de pelo menos uma linguagem de programação. Portanto, o programador experiente pode ser tanto um aluno de graduação que já cursou as disciplinas básicas de programação como um profissional já graduado e com experiência no mercado de trabalho. Considerando nossa classificação, certamente não haverá uma separação nítida na fronteira novato-experiente, da mesma forma que haverá entre os programadores experientes aqueles menos ou mais experientes. Mesmo assim, esta classificação ainda é útil para explicar as dificuldades mais freqüentemente encontradas no ensino de programação.

Com os novatos, o bom professor terá a responsabilidade de apresentar os fundamentos da programação de maneira a formar uma base sólida para conceitos subseqüentes e o revés de lidar com a dificuldade deles perante o novo. Com os mais experientes, a facilidade de mencionar conceitos de forma implícita contrastará com as dificuldades impostas por barreiras freqüentemente estabelecidas por paradigmas e vícios associados a estilos de linguagens de programação específicas.

Ao longo da discussão que se segue, citaremos algumas dificuldades freqüentemente encontradas no ensino de programação, ligadas tanto ao planejamento quanto à representação. Alguns aspectos do ensino de programação deverão gozar de diferenças de abordagem quando aplicados a programadores novatos e a programadores experientes.

4.3.1. Dificuldades ligadas ao planejamento: a solução de problemas

Com base na conceituação que colocamos para a atividade de programação e considerando seus vários processos cognitivos, podemos categorizar o profissional que exerce a atividade de programação como uma pessoa que, a partir de um problema bem definido, é capaz de

desenvolver uma solução e implementá-la em um programa. Considerando que a etapa de *planejamento* é, de fato, a solução do problema em si e que a *representação* é o problema (secundário) de se implementar a solução de acordo com uma ferramenta particular de programação, a atividade de programação pode ser vista como uma atividade de resolução de problemas e, portanto, podemos enxergar o programador como um "resolvedor de problemas".

No contexto da computação, existem basicamente três tipos de problemas: *computacionais*, *existenciais* e *transicionais* [Ambler-97]. Problemas computacionais são aqueles para os quais tudo é conhecido antecipadamente, exceto a resposta. Em particular, nós sabemos como computar a resposta e conhecemos os dados que alimentarão nossa solução. A única razão para desenvolvermos um programa é que, sem ele, a computação da resposta levaria muito tempo e seria muito complexa. Exemplos desse tipo de problema são: calcular a trajetória de um satélite, calcular a inversão de uma matriz ou simular os efeitos do aquecimento global.

Problemas existenciais são aqueles para os quais nós não sabemos se uma solução existe, ou talvez saibamos que existe uma solução, mas que é muito difícil de ser computada. Um exemplo é um jogo de xadrez, no qual é computacionalmente impossível analisar completamente *todas* as possibilidades de jogadas, pois existem muitas possibilidades e demoraria muito, mesmo para os computadores mais rápidos. Assim, mesmo que nós possamos descrever um algoritmo para avaliação, ele não seria utilizável, e daí, nós devemos tomar outra abordagem para responder se uma solução pode ser encontrada ou não. Há casos em que sabemos antecipadamente que uma solução é impossível de ser alcançada. Por exemplo: calcular os valores de $\mathbf{a, b, c} \in N$ tal que $\mathbf{a^n + b^n = c^n}$. Este programa não pode ser implementado, dado que teríamos que variar as variáveis **a, b, c** e **n** para todo o conjunto (infinito) dos números naturais N, em busca da relação indicada.

Para os problemas existenciais, a melhor abordagem é definir regras para as quais uma solução formal tem que aderir. Dadas estas regras, nós podemos então deixar um computador procurar possíveis soluções. O resultado destas procuras pode ser nenhuma solução, uma solução, ou muitas soluções.

A terceira categoria, os problemas transicionais, são aqueles problemas nos quais temos maior experiência. Os programas que implementam soluções para esses problemas são caracterizados por algoritmos conhecidos operando em uma seqüência desconhecida de eventos. Por exemplo, programas que gerenciam contabilidade bancária ou mantêm reservas para companhias aéreas. Um importante aspecto desses programas é que eles mantêm dados persistentes, isto é, mantêm dados de um dia para o outro, atualizando-os a

cada transação efetuada. O nome transicional, deriva-se do fato de que os dados sofrem transição de estados.

A categorização de problemas é importante durante o desenvolvimento de certas habilidades básicas de programação, tais como a decomposição de problemas e abstração, e contribui na identificação, por parte do programador, da abordagem mais eficaz para um problema em particular.

Uma vez categorizados, os problemas devem ser analisados em busca da melhor abordagem. Na visão de Newell e Simon (Newell, Simon apud [Hewett-96]:2), um problema qualquer é caracterizado por quatro elementos: 1) o estado inicial; 2) o estado final desejado; 3) um conjunto de operadores disponíveis para a transformação do estado inicial no estado desejado; e 4) um conjunto de restrições sobre os operadores. Um problema existe quando nós temos uma lacuna entre um estado inicial e um estado final desejado. A solução do problema, portanto, envolve a aplicação do conjunto apropriado de operadores necessários para completar uma série de transformações que irão eliminar a lacuna. Estas transformações devem ser conduzidas pelo resolvedor (programador) de maneira a não violar as condições e regras dos operadores. De fato, o método é simples, apesar de não significar necessariamente que o processo de solução dos problemas é igualmente trivial.

Como resultado de repetidas experiências sobre resolução de problemas, os resolvedores constróem um conjunto organizado de conhecimento ou informações sobre as propriedades de um tipo particular de problema e as operações ou passos necessários para resolvê-lo, o qual é usualmente referido como um "esquema para o problema". Ao longo da sua vida, o ser humano desenvolve uma grande variedade de esquemas de problemas, e um indivíduo típico monta um estoque de esquemas que é colocado em ação quando se identifica um problema já anteriormente esquematizado. Alguns esquemas se tornam tão familiares que são ativados quase que automaticamente, depois de uma breve análise. Problemas que não se encaixam nos esquemas armazenados são esquematizados e armazenados.

Tipicamente, o efeito dos esquemas para solução de problemas é ajudar a prover métodos razoavelmente eficientes para a solução de tipos de problemas freqüentemente encontrados. Entretanto, cabe aqui três observações quanto à estratégia de um bom resolvedor de problemas. A primeira é que o objetivo não é dispor de um banco de problemas particulares, mas de um banco de esquemas categorizados sobre tipos de problemas. A segunda é que a *representação* de um problema particular é um fator crucial na busca de um esquema ou esquemas que irá(ão) levar a sua solução. Quando o problema não é resolvido tão facilmente quanto o esperado, pode ser frutífero observar a adequabilidade da representação do problema e conduzir alguma mudança naquela

representação. Por último, os bons resolvedores de problemas sabem que os esquemas armazenados podem influenciar na avaliação do problema particular em análise e, portanto, sabem reconhecer quando estão frente a um problema realmente novo [Hewett-96].

4.3.2. Dificuldades ligadas à representação

A programação como um domínio de conhecimento difere de outros domínios vizinhos como matemática ou física em dois pontos. Primeiro, não há atividade intelectual cotidiana que possa formar a base para construção espontânea de modelos mentais para conceitos fundamentais para a programação tais como recursividade, variáveis ou controle de fluxo, por exemplo, em contraste com outras noções como número ou velocidade. Dessa forma, geralmente não há herança de conceitos que possa contribuir para o aprendizado de programação, considerando o estudante novato. Em segundo lugar, programação é uma atividade que opera em uma máquina física que pode não ser transparente em seu funcionamento para os estudantes.

As dificuldades encontradas por programadores novatos podem ser divididas em áreas correspondentes aos principais campos conceituais que devem ser absorvidos durante o processo de aprendizado de programação. Esta subdivisão foi feita inicialmente por Rogalski e Samurçay [Rogalski-90] apenas para fins de legibilidade, uma vez que os programadores novatos sofrem de dificuldades bem maiores do que simples confusões ou erros isolados, ou seja, todos os erros agem em conjunção uns com os outros e podem ter múltiplas causas em função do contexto do problema.

4.3.2.1. Representação conceitual do computador

Quando envolvidos nos processos de projetar, entender ou depurar um programa ou solução, programadores experientes fazem necessariamente referência aos seus conhecimentos dos sistemas que estão por trás das linguagens de programação e das ferramentas disponíveis nesses sistemas. Eles têm a habilidade de se mover de um sistema para outro e mudar suas representações do problema para se adaptar às novas restrições e aos novos recursos disponíveis. Este conhecimento se estende para além das regras de operação dos sistemas e alcançam a representação de todo o sistema (sistema operacional, editor, comandos, filosofia, dispositivos de entrada e saída e outros). Alguns autores chamam esta representação de *notional machine* ou *mecanismo perceptivo* (Taylor, du Boulay apud [Rogalsky-90]:162).

Existem três dificuldades básicas encontradas por programadores novatos em relação ao estabelecimento das suas representações conceituais. A primeira é relacionada à construção do *mecanismo perceptivo*, explicado anteriormente.

A segunda dificuldade envolve o entendimento do dispositivo interno que executa as instruções passadas pelo programador e da existência de entidades virtuais que simulam coisas que não têm identidade física (exemplo: memória virtual, variáveis, arquivos, apontadores). Esta dificuldade também tem uma relação com a proximidade ou distância do controle por parte do programador. Por exemplo, as linguagens de programação Assembly e Prolog são extremos de proximidade e distância do controle, respectivamente. Ou seja, em Assembly, o programador tem virtualmente o controle total dos dispositivos, o que a caracteriza como uma linguagem de baixo nível, enquanto que em Prolog o programador trabalha a uma grande distância da máquina física, o que a caracteriza como uma linguagem de alto nível. Quanto mais próximo for o controle e maior for o número de entidades virtuais envolvidas, maior é a complexidade para o novato.

A terceira dificuldade conceitual diz respeito à filosofia das ferramentas de programação, ou seja, ao paradigma de programação inerente a cada linguagem de programação. A conceitualização da "máquina Pascal", por exemplo, é diferente da "máquina Prolog", que é diferente da "máquina Lisp", que por sua vez é diferente da "máquina C++" e assim por diante.

Programas em Pascal descrevem *como* o resultado deve ser computado. Programas Prolog indicam *o que é verdadeiro* na solução do problema, deixando que o computador encontre a seqüência de passos para produzir o resultado. Para a "máquina Lisp" a solução é representada por uma cadeia de funções, cada uma construindo soluções intermediárias entre entradas e saídas, até que a ação da programação seja progressivamente a construção de bibliotecas de funções. Na "máquina C++", os objetos envolvidos no problema são modelados, seu comportamento e propriedades são descritos, o relacionamento entre eles é estabelecido e a solução é programada em função de como os objetos se relacionam, inclusive combinando alguns conceitos de outras máquinas anteriores.

O fato de que a programação pode ser conceitualizada como a descrição de comandos (paradigma imperativo), descrição de funções (paradigma funcional), descrição de relações lógicas (paradigma lógico ou relacional), descrição de objetos (paradigma de orientação a objetos) e, em alguns casos, a combinação de mais de um tipo de descrição, pode causar para um programador novato desde pequenas confusões durante a tentativa de identificar o que é controlado pela própria linguagem e o que fica para o programador, até grandes dificuldades relacionadas com a representação da solução estabelecida.

4.3.2.2. Estruturas de controle

A principal característica das estruturas de controle é o fato de que elas rompem com a linearidade de um texto de programa. Na maioria dos estudos sobre as dificuldades na

aprendizagem de programação, a abordagem procedural (linguagens imperativas) domina, embora alguns também enfoquem outros paradigmas como lógico, funcional e orientado a objetos [Ramalingam-97], [Rogalski-90], [Good-97a]. Considerando o comando incondicional (go to), os comandos condicionais (if-then-else; case-of), as interações (while-do; do-until; for-do) e a recursividade como principais pontos no entendimento das estruturas de controle que regem a programação, observa-se principalmente três dificuldades.

A primeira delas diz respeito ao entendimento completo das expressões lógicas que regem as estruturas de controles e à habilidade no uso dos conectores AND, OR, NOT.

A segunda se relaciona com a profundidade dos comandos condicionais aninhados, regras de escopo e agrupamento de blocos de comandos. Quanto maior for a profundidade do aninhamento maior será a dificuldade de percepção, para o novato, tanto no processo de composição como no de compreensão (vide Figura 27, pág. 71). Esta dificuldade se relaciona também com a imagem mental do processo de recursividade, suas regras de escopo, a compreensão das chamadas pendentes empilhadas e, principalmente o estabelecimento da condição de parada. Observe que estamos incluindo o processo de compreensão na avaliação das dificuldades envolvidas no aprendizado de programação, embora a atividade de manutenção e correção de programas não seja objeto do ensino introdutório de programação. Isso se deve ao fato de que, embora a maior parte do ensino de programação esteja ligada ao processo de composição, normalmente aceita-se como boa prática pedagógica a avaliação de trechos de código já escritos, para avaliação de boas ou más práticas ligadas à programação. Portanto, quando o aluno depara-se com um código já escrito, ele estará acionando mecanismos cognitivos ligados à compreensão.

A terceira dificuldade diz respeito à conceitualização das estruturas de controle quando da mudança do paradigma de programação. Por exemplo, não há comandos condicionais explícitos em Prolog e o tratamento condicional se dá através de condições lógicas (conjunções, negações etc) e controle de *backtracking*. Seja qual for o primeiro paradigma ao qual o novato tenha sido apresentado, o remapeamento de conceitos para um segundo paradigma apresenta uma dificuldade que, idealmente, necessita ser acompanhada por um orientador ou facilitador.

4.3.2.3. Variáveis, estrutura de dados e representação

Em um programa, as variáveis são o espelho dos objetos ou entidades que elas representam e, em qualquer que seja a linguagem de programação, elas desempenham um papel fundamental.

Um dos fatores chave para a aquisição do conhecimento sobre programação é o entendimento do papel funcional das entidades na solução e os sucessivos valores que elas assumirão ao longo da execução do programa. Isso envolve o entendimento da conexão entre funções e variáveis (ou procedimentos e variáveis, como nas linguagens imperativas, ou predicados e variáveis, como nas linguagens lógicas, ou de objetos e propriedades, como nas linguagens orientadas a objeto).

Do ponto de vista do programador, as variáveis podem ser externas ou internas. As variáveis externas são aquelas controladas pelo usuário, isto é, usadas para entrada e saída explícitas de dados. As internas são aquelas controladas pelo programador e produzem valores intermediários. Erros acontecem muito comumente quando o programador permite que uma variável exerça mais de um papel [Spoher-85].

Muitas dificuldades com variáveis são relacionadas ao escopo onde ocorrem, o tempo de vida, passagens de parâmetro por valor e por referência. Mesmo considerando que o ensino de programação atualmente estimule a construção de programas estruturados e modulares (ex. uso de procedimentos e funções, no caso de paradigma imperativo, e de métodos, propriedades e comunicação entre objetos, no caso da orientação a objetos), algumas dificuldades são freqüentemente relatadas por professores quando eles tentam esclarecer as diferenças entre variáveis locais e globais.

Outro problema conceitual ocorre com o uso de variáveis em programas recursivos em linguagens procedurais. Neste caso, a variável precisa ser vista em função da execução das interações recursivas, que envolvem os sucessivos contextos gerados pela ativação das chamadas recursivas. Neste estágio, o conceito algébrico de variável (tal como a variável x em $3x + 2 = 6$, por exemplo) pode ser transferido da matemática ou da física e falha como um modelo mental comparativo [Rogalski-90].

Um problema similar pode ser visto em Prolog entre as definições lógicas e os efetivos valores das variáveis e funções, quando os novatos encontram dificuldades em entender que a função específica definida em uma cláusula é determinada pela instanciação das entradas. Como exemplo da dificuldade que os novatos têm com o uso de variáveis em linguagens lógicas, considere os dois trechos de programa construídos para computar o maior valor entre dois valores fornecidos (vide Figura 29). O trecho em Prolog define a função `maior_que`, que avalia o maior valor entre dois valores fornecidos. Em uma linguagem imperativa, tal como Pascal, o programador lidaria com as variáveis de outra forma, comparando-as e controlando-as em maior detalhe, até alcançar explicitamente o resultado.

Prolog	Pascal

...
x maior_que(x,x)
x maior_que(x,y) if y less x
y maior_que(x,y) if x less y
...
	...
	readln(x, y);
	if x > y then write(x)
	else write(y);
	...

Figura 29 - Comparando o uso de variáveis em Prolog e em Pascal

O problema está em entender que o programa Prolog descreve *o que* é verdadeiro na solução do problema, em contraposição às linguagens imperativas, que descrevem *como* o resultado é computado. Essas diferenças de representação conceitual confundem o aluno e necessitam de esclarecimentos e comparações explícitas.

4.3.3. Ensinando programação para novatos e para experientes

Programadores novatos e experientes diferem em vários aspectos da resolução de problemas, notadamente na forma como representam problemas, nas estratégias que eles escolhem para tentar resolver um problema e na maneira como seus conhecimentos sobre domínios de problemas são organizados. Vejamos exemplos de algumas tendências entre novatos e experientes, relatados por Tom Ormerod [Ormerod-90].

a) os novatos categorizam problemas de acordo com características superficiais, tal como a área de aplicação, enquanto que os experientes categorizam problemas por algoritmos;

b) os novatos tendem a resolver problemas partindo da meta do problema, enquanto que os experientes partem dos dados para a meta;

c) os novatos escrevem o programa linha-a-linha, até alcançar uma solução monolítica, enquanto que os experientes dividem a meta do programa em unidades modulares.

Outro estudo, conduzido por Vikky Fix *et al* [Fix-93], concluiu que existem algumas diferenças na forma como programadores experientes e novatos estabelecem representações mentais durante o processo de programação.

Os mais experientes elaboram modelos mentais mais aprimorados que alcançam antecipadamente vários aspectos do problema envolvido, fato que é atribuído às diferenças de conhecimento e à prática. Observações de outros autores (Adelson, Larkin apud [Petre-97]:111) constatam os seguintes fatos acerca dos programadores experientes:

a) eles tendem a empreender mais tempo avaliando o problema e planejando a solução;

b) são mais aptos para formularem uma visão geral do problema, mas depois disso, despendem muito mais tempo no entendimento e desenvolvimento de suas representações;

c) formam um modelo conceitual detalhado e tendem a incorporar entidades abstratas, ao invés de objetos específicos do problema estabelecido;

d) seus modelos acomodam vários níveis de observação do problema e permitem simulações mentais.

Portanto, os mais experientes empreendem mais tempo construindo e explorando estruturas nas suas mentes, antes de postularem suas representações externas [Petre-97].

A existência dessas diferenças sugerem que a abordagem pedagógica a se adotar no ensino de programação para novatos deva ser diferente daquela para os já experientes. Além disso, a comparação entre novatos e experientes é importante porque as diferenças encontradas podem definir algumas contribuições quanto às estratégias usadas para se ensinar para novatos e para programadores experientes, bem como para estabelecer os objetivos a alcançar em cada uma dessas tarefas.

Ensinar para experientes é uma tarefa voltada para a adaptação de estilo, apresentação de estruturas sintáticas e semânticas, onde é possível pressupor a experiência do aluno e a posse de um conjunto de soluções estereotipadas para uma gama de problemas, bem como das estratégias para adaptá-las, compô-las e coordená-las. Em outras palavras, eles são capazes de ver e instanciar um problema em função de algum outro já visto anteriormente [Soloway-86]. Por outro lado, ensinar para novatos é ensinar a construir mecanismos orientados para a solução de problemas, uma tarefa que exige do aluno assimilação de novos conceitos e a habilidade de usar esses conceitos para resolver problemas quando eles se apresentam em diferentes contextos. Ou seja, o aluno precisa ser apresentado de forma explícita aos mecanismos, a definição de objetivos, aos planos para abordagem do problema, além das regras de implementação usando uma linguagem de programação.

4.3.4. Ensinar a programar X ensinar uma linguagem: o papel da linguagem no ensino de programação

Não existe um automóvel bom o suficiente para garantir que todos os que o dirijam o façam com perícia. Da mesma forma, não existe uma linguagem de programação que, por si mesma, induza programadores a construir boas soluções ou escrever um bom código. "Não existe e nunca existirá uma linguagem de programação na qual seja difícil escrever péssimos programas" (Flon apud [Petre-90]:107). Petre cita, ironicamente, que todos

conhecemos algum programador que é capaz de escrever um programa em FORTRAN, seja qual for a linguagem que esteja usando.

Na verdade, a linguagem de programação tanto pode facilitar como pode obstruir na expressão da solução, mas a linguagem não influencia fortemente a natureza da estratégia da solução. Em geral, a linguagem é uma ferramenta e, como tal, facilita a tarefa para a qual foi projetada. Mas o resultado depende da intenção e da perícia de quem a maneja.

Também no ensino de programação, a linguagem usada pode ser um elemento facilitador ou um elemento obstrutor. Há casos em que a linguagem de programação escolhida impõe alguma sobrecarga cognitiva aos alunos novatos, pela dificuldade que eles têm em separar a semântica e a sintaxe.

Um experimento conduzido por Walsh *et al* mostrou que a sintaxe da linguagem de programação pode ser um elemento de obstrução para o processo de representação da solução. O experimento envolveu alguns testes aplicados a 28 alunos novatos, sendo que em um dos testes propunha-se um problema para ser solucionado usando um computador, enquanto eram observados durante todo o processo de construção e depuração de um programa em Pascal. Apenas 40% deles chegaram a um programa que funcionava no computador, logicamente correto ou não. O restante dos alunos ficaram tão presos em erros de sintaxe produzidos pelo compilador que perderam-se dos seus objetivos [Walsh-89].

Se considerarmos que isso se deu com o uso de uma linguagem amplamente reconhecida como adequada para o ensino de programação, podemos concluir que algo semelhante ou pior pode estar acontecendo com o uso de outras linguagens consideradas menos adequadas ao ensino de programação para novatos, como C, FORTRAN, Clipper, Prolog e outras (vide Tabela 1, pág. 24, para uma referência das linguagens usadas no ensino de programação). Infelizmente, a experiência mostra que o tempo despendido pelo professor para assistir os alunos novatos com erros de sintaxe é considerado alto [Martin-96]. Como se não bastasse todas as dificuldades cognitivas, já discutidas anteriormente, o programador novato ainda se vê às voltas com uma barreira adicional que é, basicamente, um problema de comunicação: a sintaxe da linguagem. Uma vez que o processo de *representação* é quem fornece a realimentação para o *planejamento*, dado que através dele o programador descobre a eficácia ou não da sua solução pela observação do seu programa em funcionamento, o problema da sintaxe da linguagem acaba sendo um obstrutor para todo o processo, frustrando o aluno. Nesses casos, muito tempo da aula recai sobre a parte menos importante – a codificação – enquanto que o máximo do tempo deveria estar voltado para todo o processo, com ênfase na solução do problema.

Baseado nessas observações e considerando o ponto de vista pedagógico, a escolha da linguagem de programação para o ensino de introdução à programação deverá levar em consideração algumas características:

- A linguagem deve ser clara e direta, para que a lógica do programa seja facilmente seguida a partir de um código já escrito. Sua sintaxe deve impor a menor sobrecarga cognitiva possível, para dar fluência aos demais elementos cognitivos envolvidos na programação.
- Deve cobrir a implementação de todo o conjunto de assuntos selecionados para a disciplina na qual está sendo usada, para permitir um uso integrado e coerente, evitando o uso de uma linguagem adicional para representação de tópicos não cobertos.
- Deve facilitar o aprendizado de conceitos de alto nível e depois dos detalhes de baixo nível. Ou seja, a linguagem deve ser capaz de implementar as funções ou procedimentos de alto nível apresentados.
- Deve ser escalável para bem além do que será ensinado nas aulas, já que os alunos precisam de espaço para experimentação individual, de acordo com suas necessidades e possibilidades. É difícil para o professor explicar as limitações da linguagem para um aluno que está tentando encontrar uma forma melhor de implementar algo.
- Deve permitir experiências aplicáveis no âmbito profissional. Linguagens que só são usadas para pequenos problemas em sala de aula, mas não podem gerar uma aplicação maior, podem tolher o aluno.
- A implementação da linguagem deve ser bem documentada. De preferência, deve ser facilmente portável ou disponível em várias plataformas, para que se possa utilizá-las sem restrições de equipamento. Algumas linguagens são feitas para funcionar em ambientes específicos (*mainframes*, *workstations* com GUI – *Graphical User Interface*), os quais são de difícil reprodução em pequenos laboratórios ou na casa do aluno.
- Deve preferencialmente dispor de um ambiente integrado para edição, depuração e execução, de boa usabilidade, evitando o uso de ferramentas adicionais.

Como já discutimos antes, o ensino de programação envolve mais do que ensinar a sintaxe de uma linguagem. Envolve também a categorização e a abordagem de problemas, especificação e refinamento de soluções, a codificação (a própria representação da solução), a depuração sintática e lógica, e a crítica do resultado, que é o programa. Nesse processo, a linguagem de programação deve ser um elemento tão transparente quanto possível. Se ela é bem escolhida, bem utilizada pelo professor e bem assimilada pelos

alunos, o processo de ensinar programação terá considerável ganho, visto que tanto o professor quanto o aluno poderão se concentrar mais em todas as demais atividades da programação, além da tarefa de codificação.

O ensino de programação para novatos deve levar em consideração que o papel da linguagem de programação pode não ser o menos importante, mas certamente não é o mais importante. Ao ensinarmos ao aluno todo o processo de programação, estaremos tacitamente contribuindo para que ele aprenda a aprender novas linguagens de programação.

4.4. Elementos e estratégias para melhorar o ensino de programação

Da mesma forma que em outras áreas da computação, os conceitos de programação, as técnicas e linguagens de programação evoluem rapidamente ao longo do tempo. Por outro lado, as grades curriculares são montadas de forma a contemplar várias unidades de conhecimento, mas com uma grande (e óbvia) restrição: o total de tempo deve ser finito. Como não há uma maneira de acomodar a transferência de conhecimentos que evoluem ao longo de toda uma vida dentro de alguns anos, o tempo que dura atualmente uma graduação, é necessário investir em ações que favoreçam a incorporação, por parte do aluno, do aprendizado como um elemento importante da sua vida profissional. É preciso capacitar o futuro profissional para continuar aprendendo.

A aprendizagem continuada é um requisito fundamental para a sobrevivência do profissional após a sua desvinculação com instituições tradicionais de ensino. A aprendizagem continuada é o processo de aprender sempre e estar sempre em condições para aprender. Trata-se menos de uma aptidão individual e mais de um comportamento do profissional, associado a um conjunto de ações da sociedade (instituições de ensino, profissionais, mercado de trabalho) para favorecer a busca pela atualização profissional, seja com o retorno temporário à academia, requalificação na própria empresa, ensino não-presencial ou de outra forma.

As instituições de ensino, por sua vez, podem contribuir na formação dos alunos nesse sentido, ajudando-os a tornarem-se "resolvedores de problemas", a reconhecerem e monitorarem suas abordagens sobre tarefas particulares a partir de conceitos mas gerais e a "aprenderem a aprender".

A seguir, teceremos comentários sobre alguns recursos administrativos ou pedagógicos que podem contribuir para uma melhoria no ensino voltado para a aprendizagem continuada, tanto nos aspectos gerais quanto no ensino de programação em particular.

4.4.1. Evolução curricular e estudos sobre tendências mercadológicas

A Ciência da Computação irá continuar a evoluir rapidamente e o ritmo dessa mudança exerce e exercerá enorme pressão sobre o sistema educacional, de maneira que os currículos dos cursos devem mudar com certa freqüência. A abordagem do ensino de programação para novatos deve ter enfoque diferenciado em relação ao ensino para alunos com experiência. A linguagem escolhida como "veículo" para o ensino de programação a novatos deve ser simples tanto em sintaxe quanto em semântica, enquanto que outras linguagens mais completas poderão dar aos alunos mais experientes a visão do que o mercado requer. Um bom planejamento deverá deixar margem para reformulações curriculares mais freqüentes, sem prejuízos didáticos ou elevados custos de tempo e financeiros, causados pela manutenção de currículos paralelos.

Os mecanismos e a freqüência das evoluções curriculares variam entre algumas universidades ao redor do mundo. Na Uppsala University (Suécia), University of the Witwatersrand (África do Sul), Deakin University (Austrália) e em algumas brasileiras como UFRJ, UFSC, UFRGS, UFPB, PUC-Campinas, por exemplo, os currículos sofrem revisões maiores[8] a cada quatro ou cinco anos, com revisões menores feitas pelo coordenador de tempos em tempos, conforme necessário [Daniels-98]. A Universidade Aberta de Israel (The Open University of Israel), por exemplo, faz revisões maiores a cada sete anos. Todas as revisões levam em conta as exigências locais das instituições de certificação, comparações com os programas de outras universidades do mesmo país, recomendações de organismos como a SBC [Bigonha-96], e tendências mundiais, tais como as indicadas pela ACM e IEEE [Denning-91], [Tucker-91].

Uma estratégia para promover a atualização curricular é a criação de um Comitê de Revisão Curricular - CRC, com escopo de atuação departamental, em caráter consultivo, composto por professores de várias áreas de conhecimento, consultores internos e externos e chefiado pelo coordenador do curso ou por um professor sênior. O CRC teria como algumas atribuições:

- Avaliar tendências mercadológicas regionais e globais. Instituir e coordenar grupos de trabalho sobre novas tecnologias e tendências. Avaliar recomendações de organismos oficiais ligados à área, tais como: Comissão de Especialistas do MEC, SBC, IEEE e outros.

[8] Entendemos por revisões maiores as reformas curriculares mais amplas, que atingem os pré-requisitos, sequenciamento de disciplinas, remoção e inserção de novas disciplinas. Revisões menores são aquelas que atingem conteúdo curricular, carga horária ou metodologia de disciplinas isoladas, sem alteração da grade curricular.

- Estabelecer políticas de intercâmbio de informações com outras instituições e com organizações representativas dos segmentos de mercado atendidos. Criar ou adotar metodologias para pesquisas de opinião e prospecção de mercado.
- Estabelecer critérios e ações para adoção das tendências de acordo com o perfil do curso e do egresso. O comitê avaliaria: a) em que grau uma certa tendência afeta o curso em questão e, caso necessário, quais ações deveriam ser empreendidas para incorporá-la no currículo; b) quais disciplinas deveriam ser atualizadas e quais as mudanças necessárias.
- Usar avaliações feitas pelos estudantes como indicadores de necessidades e avaliar mudanças necessárias.
- Planejar a migração por parte dos alunos de currículos já em andamento para novos currículos, por ocasião de mudanças.
- Propor a criação, extinção ou reformulação de cursos.

O CRC submeteria suas conclusões ao departamento para discussão e delineamento das ações administrativas e estratégias para implantação de mudanças, quando cabidas.

4.4.2. Acompanhamento do egresso

O sucesso profissional do egresso é o objetivo principal de uma instituição de ensino, e tudo quanto é feito por ela visa a preparação do aluno em direção ao perfil escolhido para o egresso. Centrado nesse ponto de vista, o egresso não só deve ser observado como deve-se fazer dele um parceiro na constante (re)construção das ações que visam a qualidade da formação dos egressos subseqüentes, formando um ciclo realimentado. Portanto, a instituição deve, na medida do possível, realizar freqüentes pesquisas, visando acompanhar as atividades dos egressos, possíveis desvios, casos de fracasso e sucesso, realimentando o processo de evolução curricular.

Como mecanismo para manter-se atualizado sobre as opiniões atividades dos egressos, sugere-se que a instituição de ensino mantenha um banco de dados com informações básicas sobre os egressos (nome, endereço atualizado, e-mail, áreas de interesse, empresas onde trabalhou, cursos que freqüentou, atividades desenvolvidas, etc) e realize freqüentes pesquisas de opinião e chamadas para atualização de dados.

Como mecanismo de comunicação, poderia ser disponibilizada uma "home page" contendo informações, formulários eletrônicos de cadastramento, opções de consulta e alteração de dados, bem como a disponibilização de um endereço eletrônico, em caráter permanente. Outras formas de comunicação também poderiam estar disponíveis, como fax e correio postal.

Os recém-graduados teriam seus dados automaticamente inseridos no banco de dados. Quanto aos egressos formados anteriormente, uma campanha de divulgação poderia ser feita convidando-os a cadastrarem-se.

Como atrativo para que o egresso mantenha-se em contato permanente, alguns serviços poderiam ser oferecidos, como o envio de informações sobre a profissão, oportunidades de emprego, divulgação de eventos locais à região do egresso, consulta de informações atualizadas sobre colegas contemporâneos, entre outros. O próprio banco de dados poderia constituir um "banco de competências", o qual estaria disponível para empregadores e para comunidade em geral para consultas sobre profissionais que atendem a um determinado perfil desejado.

4.4.3. Requalificação

A pesquisa 2 apontou que a participação em "cursos de atualização promovidos por universidades" é um recurso pouco utilizado pelos profissionais do mercado. Aliás, o menos utilizado entre todos os recursos. Este fato nos leva a crer que tanto a instituição de ensino quanto o profissional acreditam que a formação do profissional acaba junto com sua graduação.

Como indicado através da pesquisa 2, o tempo decorrido desde a graduação é um fator que contribui diretamente para a dificuldade de atualização tecnológica do profissional. Sendo assim, a requalificação é um importante componente da filosofia de aprendizagem continuada e ela contribui para manter o profissional próximo das inovações tecnológicas, além de aproximar as instituições da prática dos profissionais.

Como alternativa para solucionar tal problema, as instituições de ensino poderiam instituir cursos de curta duração com caráter de atualização para permitir o retorno freqüente do profissional à academia para requalificação. Os temas seriam específicos, escolhidos a partir de pesquisas de mercado. Os cursos poderiam ser ministrados presencialmente, em horários alternativos ao horário comercial (à noite, aos sábados) ou poderiam ser não-presenciais, assíncronos (os alunos iniciariam e concluiriam o curso em pontos diferentes no tempo) e variáveis (os alunos escolheriam a duração do curso, baseados no tempo que poderiam dispor).

Estes cursos poderiam introduzir metodologias experimentais de ensino (ex. ênfase em equipes e no trabalho colaborativo, uso de tutoriais, etc) que, criteriosamente avaliados e adaptados, poderiam ser introduzidos nos cursos de graduação.

4.4.4. Enfoque prático X fundamentação teórica

Diferentes instituições têm diferentes prioridades e objetivos, da mesma forma que os cursos que mantêm. Alguns cursos são mais dirigidos para as necessidades do mercado enquanto que outros direcionam seus esforços para os objetivos mais gerais da educação científica. Devido a estas diferenças, não há um modelo curricular geral que se aplique a todos os cursos da área de computação. Entretanto, também não há uma fronteira tão clara (e muito menos rígida) entre o mercado, a academia e a comunidade científica que justifique a existência de currículos onde a fundamentação teórico-científica e o desenvolvimento de habilidades práticas sejam excludentes.

Como constatamos na pesquisa 2, 7% dos profissionais que declararam ter dificuldade de aprender novos conceitos ou uma nova linguagem de programação incluiu "o enfoque excessivamente acadêmico da graduação e a pouca orientação para o mercado" como um dos motivos que contribuíram para tal dificuldade. De fato, muitas instituições de ensino levam pouco em consideração as necessidades do mercado, o qual absorve a maioria dos graduados nas áreas de computação [Tucker-96b]. Entretanto, não se deve pregar que o correto seria um enfoque muito mercadológico ou tecnológico, pois sabemos que isso levaria a conseqüências de longo prazo ainda piores, uma vez que o mercado é dinâmico e sempre incorpora novos conceitos e novas tecnologias, enquanto que o profissional, sem a suficiente fundamentação teórica, sofreria para acompanhar as mudanças. Mas quem pode garantir que o enfoque muito acadêmico ou científico seria melhor? Não há certo ou errado, pois o que importa é o perfil do egresso que desejamos formar e o nicho de mercado que irá absorvê-lo.

O que propomos, portanto, é que as instituições de ensino e o mercado de trabalho devam desenvolver uma relação simbiótica, na qual a troca de experiências seja uma constante fonte de enriquecimento mútuo, seja qual for o perfil do egresso ou o enfoque do currículo. A proximidade com o mercado traz benefícios tanto para o cientista quanto para o tecnicista, no momento em que ambos passam a ter uma visão ampliada da aplicabilidade dos conceitos apresentados. Os currículos devem ser ajustados considerando não somente uma forte base teórica, mas também as práticas e as tendências do mercado, em função do perfil do egresso, sem que isso se traduza em prejuízos pedagógicos para o processo de ensino.

4.4.5. Laboratório: solução de problemas, times, liderança

Programação é uma matéria essencialmente de laboratório. O laboratório é um componente fundamental para o desenvolvimento dos perfis desejados (solução de problemas) e o grande facilitador para a experimentação, formulação e teste imediato de hipóteses, e da

observação e análise do comportamento desses componentes. O enfoque deve, entretanto, transcender a experiência de construir um programa, e chegar a como representar o problema, como descrevê-lo e como solucioná-lo usando um conjunto apropriado de ferramentas. Para tanto, o professor deve antecipadamente preparar atividades bem planejadas para exercitar os perfis desejados.

Muito do trabalho de um profissional de computação se dá em grupos ou times. O laboratório deve também ser usado para dar lugar ao trabalho em equipe, com ênfase no trabalho colaborativo, na relação aluno-aluno, facilitado e enriquecido pelo professor.

Os alunos devem ser apresentados às situações reais que ocorrem nos projetos em equipe e às técnicas de comunicação eficiente entre os membros, como fazer reuniões eficientes, treinar exposição de idéias em público, técnicas de elaboração de relatórios e projetos, gerenciamento de projetos, gerenciamento de pessoas, liderança. O professor deve estimular o papel dos membros e fazer rodízios entre os diversos papéis exercidos dentro de um grupo de trabalho.

Além disso, a atividade desenvolvida em laboratório deve ser uma extensão da sala de aula, monitorada pelo professor. O laboratório pode e deve continuar livre para os alunos, mas não se pode esperar que eles construam sozinhos o perfil que queremos. É preciso interferir e ajudá-los explicitamente em suas experiências. É preciso investir mais esforço nas atividades de laboratório, principalmente quando se tratar de alunos novatos.

4.4.6. Tutoriais

A pesquisa 1 revelou que grande parte dos docentes acreditam na eficácia didática dos tutoriais de software no ensino de programação (vide Tabela 2, item I), enquanto que a pesquisa 2 revela que os profissionais se utilizam pouco desse recurso (vide Tabela 7, item J). Embora a abordagem difira entre vários tipos de software educacionais avaliados, estudos comprovam que os programas CAI – *Computer Aided Instruction* (ou simplesmente "tutoriais inteligentes"), quando bem construídos, melhoram o aprendizado dos alunos [Papert-80]. Estes softwares são particularmente úteis na realização de exercícios, no desenvolvimento do raciocínio lógico e do pensamento dedutivo e indutivo [Bass-84]. Ambientes de ensino por computador podem abordar unidades de conhecimento menores (ex. "conceitos de recursividade em programação" ou "representação visual de estruturas de dados em ação") ou podem ser mais amplos (ex. "curso de programação orientada a objetos").

Alguns pontos que podem ser destacados na avaliação de um bom software instrucional são: a) uso de hipermídia – sons, imagens, movimento de componentes explorando

variados aspectos da percepção humana; b) boa navegabilidade – para reduzir a sobrecarga cognitiva do próprio sistema sobre o conhecimento apresentado; c) interatividade – para permitir, entre outras coisas, marcar pontos que não entendeu, registrar observações pessoais durante o processo, imprimir observações ou figuras, salvar ponto atual da instrução, etc; e d) estruturação hierárquica – tópicos de alto nível contendo informações mais importantes, tornando-se progressivamente específica e detalhada [Recker-92].

Além disso, um bom software instrucional deve respeitar os limites individuais do aluno, seu perfil e seu ritmo de aprendizado, evitando avaliá-lo diretamente através de escores. Além disso, sugere-se avaliar e validar este tipo de ferramenta em conjunto com outros professores, considerando as condições e o contexto onde será aplicado.

O uso do software instrucional deve ser, sempre que possível, associado a discussões com o professor ou com o grupo, valorizando a interação professor-aluno, considerada a mais eficaz modalidade de ensino [Soloway-96].

5. Conclusão

A discussão de estratégias para a aprendizagem continuada do profissional de programação envolve o conhecimento dos mecanismos cognitivos básicos envolvidos no aprendizado e na prática da atividade de programação. Como ponto de partida para essa discussão, buscamos também ampliar nosso conhecimento sobre três aspectos fundamentais na vida do profissional de programação: sua formação, sua prática e os mecanismos usados para sua atualização técnica. Para tanto, foram realizadas duas pesquisas: a primeira sobre o ensino de programação nas instituições de ensino superior brasileiras (pesquisa 1), e a segunda sobre a prática do profissional no mercado de trabalho, suas dificuldades de adaptação e os mecanismos usados para sua atualização (pesquisa 2).

Da pesquisa 1, observa-se uma predominância do paradigma "imperativo" como sendo o primeiro paradigma de programação apresentado aos alunos. A preferência pela adoção desse paradigma é visivelmente decrescente à medida em que o aluno avança na grade curricular e outros paradigmas são abordados. O paradigma "objeto" apresenta o comportamento inverso, tendo enfoque crescente ao longo do curso. Os paradigmas "funcional" e "lógico" são pouco enfocados em disciplinas introdutórias e apenas são abordados depois dos paradigmas "imperativo" e "objeto". As linguagens do paradigma "concorrente" não são geralmente abordadas nos currículos brasileiros, talvez por não ter tanto respaldo no mercado de trabalho, que de fato não o utiliza na prática, como pode ser inferido da pesquisa 2.

Por ter sido realizada pela primeira vez, nossa pesquisa sobre o ensino de programação nas IES brasileiras não contabiliza informações relativas aos anos anteriores, o que não nos permite estabelecer comparações quanto à evolução do uso das linguagens de programação nos currículos. Entretanto, foi possível perceber nas instituições pesquisadas a gestação de propostas de mudanças na forma de ensinar programação e a disposição em avaliar a eficácia das mudanças pioneiras já realizadas por outras instituições, principalmente na adoção do paradigma "objeto" em substituição ao "imperativo". Espera-se que pesquisas semelhantes possam ser realizadas futuramente visando estabelecer comparações, indicar tendências e avaliar a implantação de metodologias alternativas para o ensino de programação, incluindo a avaliação prática do uso de outros paradigmas nas disciplinas introdutórias de programação, visando fornecer subsídios para os professores.

Foi possível caracterizar, baseado na compilação de vários trabalhos, que a atividade de programação envolve basicamente dois aspectos: *planejamento* e *representação*. No primeiro, um esforço cognitivo é empreendido pelo programador para estabelecer uma abordagem adequada ao problema, gerando-se uma especificação da solução, ainda no

domínio do mundo real. No segundo, o programador representa a solução para o problema usando-se uma ferramenta computacional – a linguagem de programação. Ou seja, ele constrói um mapeamento da solução estabelecida, gerando um texto que pode ser interpretado sem dubialidades: o programa. A programação envolve, ainda, a conexão de sub-tarefas ligadas a vários domínios de conhecimento, tais como o domínio de conhecimento específico do problema, a elaboração de estratégias (planejamento, algoritmos, métodos), a codificação (técnicas de programação, linguagens) e a documentação. Além disso, outros aspectos relacionados ao trabalho colaborativo, isto é, em equipes, são relevantes na atividade prática do programador e para permitir a aprendizagem continuada.

A atividade de programação se estende para além da atividade de codificação e ensinar a programar requer do professor o conhecimento das principais dificuldades enfrentadas pelos alunos, que variam desde a correta abordagem e categorização do problema até os aspectos ligados à representação (entendimento dos dispositivos computacionais, sintaxe da linguagem, estruturas de controle, estruturas de dados, técnicas etc). No que se refere ao ensino de programação, o professor também deverá ter sensibilidade para abordar de maneira diferente alunos novatos e experientes, os quais apresentam dificuldades cognitivas de ordem e magnitude diferentes.

É possível concluir, da pesquisa 1, que a maioria dos professores concorda que é necessário estabelecer uma separação mais clara entre o ensino de *programação* e o ensino de *linguagens* de programação para o alcance de melhores resultados. De fato, a linguagem de programação, embora importante, é apenas um componente na atividade de programação. O enfoque excessivo na linguagem pode descaracterizar a amplitude do processo envolvido na programação e eventuais mudanças no uso de linguagens visando a busca de melhores resultados sem a devida consideração aos demais componentes envolvidos pode ser, no mínimo, inócua. Afinal, é possível se construir bons ou maus programas em qualquer linguagem. A linguagem de programação é uma ferramenta, e como tal, facilita a tarefa para a qual foi projetada, mas o resultado depende da perícia de quem a maneja. Assim sendo, fica evidente que o uso de novas linguagens e paradigmas devem levar em consideração uma possível mudança na metodologia como importante aliada para o alcance de melhores resultados.

Considerando, entretanto, que os aspectos pedagógicos e metodológicos são mais importantes do que a escolha de uma linguagem de programação particular, essa escolha deveria recair em uma linguagem simples, prática, atual e em consonância com o mercado de trabalho, apenas por uma questão de motivação para o aluno e para reduzir o tempo de adaptação ao mercado, após a conclusão da sua graduação.

Os professores também concordam que disciplinas ligadas ao ensino de programação carecem de uma padronização de conteúdos e que, embora seja óbvio que a seqüência na qual os conceitos devem ser apresentados é importante, existem discrepâncias no entendimento quanto à seqüência ideal, entre os cursos de graduação. Outro ponto de convergência da maioria dos professores é que a parte prática e a parte teórica das disciplinas de programação são igualmente importantes, ou seja, nenhuma delas é mais importante que a outra.

Quanto à aprendizagem continuada dos futuros profissionais, uma revelação importante é que a maioria dos professores não acredita que, depois de graduados, seus alunos conseguirão se adaptar com facilidade às mudanças e evoluções que ocorrem na área de programação. Provavelmente há uma consciência de que há algo por completar na formação do aluno para habilitá-lo a continuar aprendendo, além das habilidades essencialmente técnicas.

Como estratégia para favorecer a aprendizagem continuada, alguns outros conhecimentos adicionais devem ser enfatizados, visando complementar e expandir as habilidades técnicas do aluno em formação. Elaboração de propostas de projeto, planejamento, trabalho em equipes (inclusive multidisciplinares), gerenciamento de pessoas, técnicas de relacionamento interpessoal, liderança, gerenciamento do tempo, apresentação de idéias em público, elaboração de relatórios de resultados e manuais do usuário etc, são exemplos de conhecimentos pouco abordados na formação do profissional, mas que fariam grande diferença qualitativa se fossem trabalhados de forma integrada nos currículos. É fundamental que as grades curriculares adotem disciplinas (ou mesclem em disciplinas já existentes) com conhecimentos que contribuam de maneira efetiva para a formação desses perfis tão importantes para manter a capacidade de aprender do aluno, futuro profissional.

Uma vez concluída sua graduação, o então profissional se vê às voltas com a adaptação ao mercado de trabalho e, passado um pouco mais de tempo, com a preocupação em se manter atualizado.

Considerando o perfil traçado pela pesquisa 2, as linguagens mais dominadas pelos profissionais são C, Clipper, Pascal, C++, Delphi (OOP), nessa ordem, mostrando que o paradigma "imperativo" também predomina como o mais dominado entre os profissionais. Foi revelado também que após a ruptura com os mecanismos formais de ensino, a atualização do profissional é predominantemente auto-didática, informal e, quando formal, o investimento financeiro é feito por conta própria. Participação em cursos de atualização promovidos por universidades e congressos é um recurso muito pouco utilizado em relação aos demais métodos de atualização.

A dificuldade em se manter atualizado atinge 44% dos profissionais e a dificuldade de aprender uma nova linguagem ou um conceito novo atinge 29%, dentre os que participaram da pesquisa. Apenas 10% e 14% declararam não ter dificuldade de atualização e de aprender uma nova linguagem, respectivamente. Apesar de pequenas variações, não foram verificadas diferenças significativas quanto à utilização de mecanismos de atualização entre os profissionais que apresentam maior e menor dificuldade. Ou seja, aparentemente não é o meio usado para se atualizar que interfere na maior ou menor dificuldade em se manter atualizado.

Não foi surpresa constatar que a dificuldade de atualização é proporcional ao tempo decorrido desde a graduação, o que reforça a necessidade de buscarmos cada vez mais meios eficientes para habilitar o profissional a se manter aprendendo de forma contínua.

Os resultados obtidos também sugerem a necessidade de um plano de evolução curricular eficiente para os cursos da área, o que pode envolver ações pedagógicas e também administrativas, tais como o acompanhamento eficiente do egresso e a avaliação de tendências de mercado como fonte de realimentação para novos currículos e a adoção de novas metodologias baseadas na solução de problemas. A prática em laboratório é um elemento que deve merecer maior enfoque nos aspectos didáticos para extrair ganhos extras a partir do trabalho colaborativo e interdisciplinar, do potencial didático e cognitivo da comunicação aluno-aluno e do trabalho em equipes, sempre com mais enfoque no processo do que no produto.

A aprendizagem continuada sugere uma mudança na forma como as instituições formadoras vêem seus ex-alunos e, em geral, os profissionais que atuam no mercado. Em geral, atualmente, o aluno é admitido para um programa de graduação, cursa disciplinas por alguns anos, recebe uma formação adequada para o cenário social, tecnológico e mercadológico vigente e, após a conclusão, é desligado do sistema acadêmico, passando a ser responsabilidade do mercado de trabalho. Com o dinamismo e a rápida evolução das tecnologias ao longo do tempo, o profissional poderá precisar de uma requalificação.

Como uma alternativa para solucionar tal problema, as instituições de ensino poderiam instituir cursos de curta duração com caráter de atualização, em adição aos atuais programas de pós-graduação, para permitir o retorno do profissional à academia para requalificação em temas específicos ou gerais, de forma presencial ou não-presencial. Um programa eficiente de acompanhamento de egressos deve ser desenvolvido e uma constante avaliação de tendências mercadológicas daria suporte à escolha de temas dos cursos, enquanto um criterioso trabalho para a elaboração de metodologias adequadas daria suporte à execução e avaliação dos resultados.

Os tutoriais de software também podem ser um excelente complemento à instrução do professor, dentro ou fora da aula e a agregação de outras tecnologias e metodologias (redes, videoconferência, material didático, avaliações simuladas e exercícios disponíveis em *home pages*), tais como o ensino à distância, são uma alternativa para a interação com o aluno durante o curso e para o desenvolvimento de programas de requalificação para profissionais.

Por fim, acreditamos que o método usado neste trabalho pode ser adaptado e estendido a outras matérias da computação e que os resultados obtidos são um bom começo para avaliar a relação do ensino e da prática da programação com a aprendizagem continuada e para lançar um pouco mais de "luz" sobre esse campo de estudo tão amplo e certamente ainda não esgotado.

Referências Bibliográficas

[ACM-68] ACM Curriculum Committee on Computer Science. *Curriculum 68: Recommendations for the undergraduate program in computer science*; Communications of the ACM, Vol 11, 3, Março 1968; Pág 151-197.

[Ambler-97] Allen Ambler; Kathy Coggins; *EECS 188: Computational Problem Solving*; Notas de aula da disciplina EECS 188, Cap. 1; The University of Kansas – Kansas, EUA; 1997; at <URL http://www.eecs.ukans.edu/~188admin/f97/chaps/intro.html>

[Astrachan-96a] Astrachan, Owen; *Education Goals and Priorities*. ACM Computing Surveys. Vol. 28, Nº 4. 1996.

[Astrachan-96b] Owen L. Astrachan. *A Computer Science Tapestry: Exploring Computer Science and Programming with C++*; New York: McGraw-Hill, 1996. Informações resumidas obtidas em <URL http://www.cs.duke.edu/~ola/book.html>

[Austing-79] Richard H. Austing, Bruce H. Barnes, Della T. Bonnette, Gerald L. Engel, Gordon Stokes; *Curriculum '78: Recommendations for the Undergraduate Program in Computer Science*; Communications of the ACM, Vol 22, 3, Março 1979; Pág. 147-166

[Bass-84] Bass, G. M., Jr., and Perkins, H. W.; *Teaching Critical Thinking Skills with CAI*; Electronic Learning 14/2 (1984): 32, 34, 96.

[Bigonha-96] Bigonha, R. S; Nunes, J. D; Salgado, A. C.; Jonathan, M.; Pádua, C. I. P. S. P.; Costa, T. S.; *Currículo de Referência para Cursos de Graduação em Computação, Versão 1996 – Parte I*, Anais do IV Workshop de Educação em Informática", Brasília – DF, 1997.

[Brooks-77] Brooks, Ruven; *Towards a Theory of the Cognitive Processes in Computer Programming*; International Journal of Man-Machine Studies; 9:737-751; 1977

[Bruce-96] Bruce, Kim; *Thoughts on Computer Science Education.* ACM Computing Surveys. Vol. 28, Nº 4. 1996.

[Butterfield-94] Butterfield, Jeff; Cooprider, Jay G.; Rathnam , Sukumar; *Resolving cognitive conflict in requirements definition: a blackboard-based model and system architecture*; SIGCPR '94. Proceedings of the 1994 computer personnel research conference on Reinventing IS: managing information technology in changing organizations, pages 105-115.

[Daniels-98] Daniels, Mats; Gal-Ezer, Judith; Sanders, Ian; Teague, G. Joy; *Teaching Computer Science: Experiences From Four Continents*; Department of Computer Systems, Uppsala University, Sweden; 1998; AT <URL http://www.docs.uu.se/~matsd/SIGCSEpaper.html>

[Denning-91] Denning, P., D. Comer, D. Gries, M. Mulder, A. Tucker, A. Turner e P. Young, "*Computing as a Discipline*.", Report of ACM Task Force on the Core of Computer Science, ACM Press, New York, 1988.

[Fix-93] Fix, Vikky; Weidenbeck, Susan; Scholtz, Jean; *Mental representations of programs by novices and experts*; CHI '93. Conference proceedings on Human factors in computing systems, pages 74-79

[Good-97a] Good, Judith.; Brna, Paul; *Explaining Programs: when talking to your mother can make you look smarter*; Human Communication Research Centre, University of Edinburgh, Scotland, UK; 1997

[Good-97b] Good, Judith.; Brna, Paul; *Information Types and Cognitive Principles in Program Comprehension: Towards Adaptable Support for Novice Visual Programmers*; Human Communication Research Centre, University of Edinburgh, Scotland, UK; 1997

[Hewett-96] Hewett, Thomas; *Cognitive factors in design: basic phenomena in human memory and problem solving*; CHI '96. on Human factors in computing systems---conference companion: Common Ground, pages 367-368

[Kereki-97] Inés Friss de Kereki; *Enseñando y Aprendendo Programación Orientada a Objetos en Los Primeros cursos de Programación: la experiencia de la Universidad ORT Uruguay*; Universidad ORT Uruguay; 1997

[Letovsky-86] Stanley Letovsky; Elliot Soloway; *Delocalized Plans and Program Comprehension*. IEEE Software, 19(3):41–48, Março 1986.

[Marconi-90] Marconi, M. de Andrade, Lakatos, E. M.; *Técnicas de Pesquisa*; Editora Atlas, São Paulo – SP, 1990

[Martin-96] Martin, Jacqueline L.; *Is Turing a better language for teaching programming than Pascal?*; B.Sc. Computing Science and Education Honours Dissertation; Department of Computing Science; University of Stirling; January 1996

[McCauley-96] McCauley, Renée A.; Manaris, Bill Z.; *Comprehensive Report on the 1996 Survey of Departments Offering CSAC/CSAB-Accredited Degree Programs*; Computer Science Department; University of Southwestern Louisiana, Lafayette, LA, 1996; at <URL http://www.cacs.usl.edu/~mccauley/survey/report1996/>

[Nunes-94] Nunes, J. D; Bigonha, R. S; Costa, T. S.; Setzer, V, W; "*Currículos de Referência da Sociedade Brasileira de Computação para Cursos de Graduação Plena em Computação*", Anais do XIV Congresso da Sociedade Brasileira de Computação, Caxambú – MG, 1994.

[Ormerod-90] Ormerod, Tom; *Human Cognition and Programming*; In: Hoc J-M., Green T., Samurçay R., Gilmore D.; Psychology of Programming; pp. 63-82; Academic Press; Orlando, Florida; 1990.

[Papert-80] Papert, Seymour; *Mindstorms: Children, Computers, and Powerful Ideas*; Basic Books; New York; 1980.

[Pennington-90] Pennington, N.; Grabowski, B.; *The Tasks of Programming*. In: Hoc J-M., Green T., Samurçay R., Gilmore D.; Psychology of Programming; pp. 45-62; Academic Press; Orlando, Florida; 1990.

[Petre-90] Petre, Marian; *Expert Programmers and Programming Languages*. In: Hoc J-M., Green T., Samurçay R., Gilmore D.; Psychology of Programming; pp. 103-115; Academic Press; Orlando, Florida; 1990.

[Petre-97] Petre, Marian; Blackwell, Allan F.; *A glimpse of mental programmer's mental imagery*; ESP'97. Papers presented at the seventh workshop on Empirical studies of programmers, pages 109-123, 1997.

[Pitkow-94] James E. Pitkow, Colleen M. Kehoe; *Results from the Third WWW User Survey*; Graphics, Visualization, & Usability Center, College of Computing, Georgia Institute of Technology, Atlanta; 1994; at <URL http://www.w3j.com/1/pitkow.107/paper/107.html>

[Proulx-91] Proulx, Vera; Rasala, Richard. *The future of Computer Science Education*. ACM Computing Surveys. Vol. 28, N° 4. 1996.

[Ramalingam-97] Ramalingam, Vennila; Wiedenbeck, Susan; *An empirical study of novice program comprehension in the imperative and object-oriented styles*; ACM Computing Surveys. Vol. 28, N° 4. 1996.

[Recker-92] Recker, M. Margaret; Pirolli, Peter; *Student strategies for learning programming from a computational environment*; Proceedings of the International Conference on Inteligent Tutoring Systems, pp 382-394; Berlin: Springer Verlag; 1992.

[Rogalski-90] Rogalski, Janine; Samurçay, Renan; *Acquisition of Programming Knowledge and Skills*. In: Hoc J-M., Green T., Samurçay R., Gilmore D.; Psychology of Programming; pp. 157-174; Academic Press; Orlando, Florida; 1990.

[Sharp-96] Helen Sharp, Mary Lynn Manns, Phil McLaughlin, Maximo Prieto, Mahesh Dodani; *Pedagogical Patterns – Successes in Teaching Object Technology*; ACM SIGPLAN Notices, Vol. 31(12), December 1996, pp. 18-21.

[Soloway-86] Soloway, Elliot; *Learning to program = learning to construct mechanisms and explanations*; Communications of ACM; Vol. 29, N° 9, pag 850-858; 1986

[Soloway-96] Soloway, Elliot; *Teachers are the key*; Communications of the ACM; Vol. 39, No. 6 (June 1996), Pages 11-14

[Spoher-85] Spoher, J. C.; Soloway, E.; Pope, E.; *A goal-plan analysis of buggy Pascal programs*; Human-Computer Interaction, 1, 163-207; 1985.

[Stacy-95] Stacy, Webb; Macmillian, Jean; *Cognitive Bias in Software Engineering*; Communications of the ACM; Vol. 38, No. 6 (June 1995), Pages 57-63.

[Tucker-91] Tucker, A. (Ed), B. Barnes, R. Aiken, K. Barker, K. Bruce, J. Cain, S. Conry, G. Engel, R. Epstein, D. Lidtke, M. Mulder, J. Rogers, E. Spafford e A. Turner, *"A Sumary of the ACM/IEEE Joint Task Force Report Computing Curricula 1991"*; Communications of ACM; Vol. 34, 6; 1991, pág. 69-84.

[Tucker-96a] Tucker, Allen B. *Crisis in Computer Science Education*. ACM Computing Surveys. Vol. 28, N° 4. 1996.

[Tucker-96b] Tucker, Allen B. (Ed); *Strategic Directions in Computer Science Education*; ACM Computing Surveys; Vol. 28, No. 4; 1996.

[Walsh-89] Walsh, T; *Rationale for and problems involved in teaching programming*; Unpublished M. Ed. Dissertation, University of Stirling, Stirling; 1989

[Watt-91] Watt, David A.; *Programming Concepts and Paradigms*; Prentice Hall; New York – 1991

[Whitelaw-95] Whitelaw, M; Weckert, J; *The Humanness of Object-Oriented Programming*; School of Information Studies, Charles Sturt University – Riverina, Wagga Wagga, Austrália; 1995; at <URL http://kcox.cityu.edu.hk/ct1995/whitela1.htm>

Anexos

Anexo A. Formulário da Pesquisa sobre o ensino de programação

Nas páginas seguintes, o formulário eletrônico da Pesquisa 1 é apresentado na íntegra, de acordo como foi mostrado para o usuário. O fluxograma apresentado na Figura 30 abaixo descreve a seqüência de páginas apresentadas ao usuário, a partir de uma página básica comum às duas pesquisas. A figura mostra também as ações executadas pelo mecanismo de controle e recepção dos dados, que efetuava a consistência e armazenamento para posterior tratamento, o qual foi baseado na tecnologia CGI – *Commom Gateway Interface.*

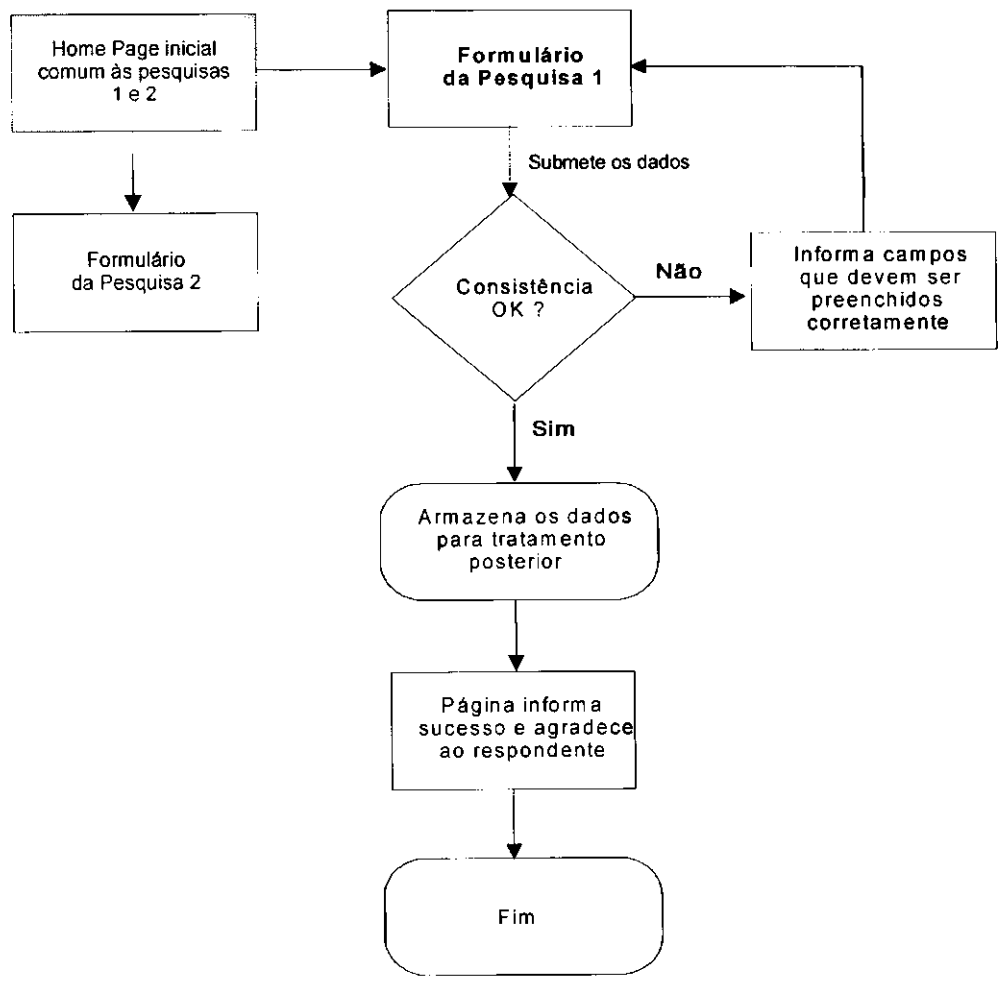

Figura 30 - Fluxo de informações do formulário da Pesquisa 1

UNIVERSIDADE FEDERAL DA PARAÍBA

O DSC - Departamento de Sistemas e Computação - da UFPB, através de alunos da pós-graduação, está desenvolvendo uma pesquisa que visa buscar alternativas metodológicas para o ensino de programação.

Para isso, desenvolvemos alguns formulários de coleta de dados para obter contribuições dos profissionais da área.

Ao responder os questionários, você estará contribuindo com sua experiência para o estudo e desenvolvimento de uma nova metodologia para o ensino de programação e se cadastrará para receber, por e-mail, os resultados da pesquisa.

Form1: Informações sobre o Ensino de Programação (clique para acessá-lo)

Quem deve responder:

- Professores que lecionem disciplinas ligadas ao ensino de programação em qualquer curso de graduação
- Professores de quaisquer disciplinas de cursos de graduação na área de informática
- Coordenadores de cursos de graduação na área de informática
- Ex-coordenadores de cursos de graduação na área de informática

---

Form2: Informações sobre a Prática dos Profissionais de Programação (clique para acessá-lo)

Quem deve responder:

- Profissionais ligados à área de programação, com qualquer formação de nível superior.
- Programadores com formação superior
- Analistas de sistemas
- Engenheiros de Software
- Gerentes de projetos de software

Para responder ou conhecer detalhes sobre esta pesquisa, clique em algum dos formulários acima.

UNIVERSIDADE FEDERAL DA PARAÍBA

Coleta de informações sobre o Ensino de Programação nos Cursos de Graduação

Sim, quero responder o questionário !

Objetivos:

(VOLTA AO TOPO) (RESPONDER O QUESTIONÁRIO)

- Este questionário foi preparado para coletar dados para uma pesquisa desenvolvida por alunos de mestrado do Curso de Pós-Graduação em Informática da UFPB. Ao respondê-lo, você estará contribuindo com sua experiência para o estudo e desenvolvimento de uma nova metodologia para o ensino de programação. Se você informar seu e-mail no questionário, teremos o prazer de enviar-lhe os resultados da pesquisa.

Quem deve responder:

(VOLTA AO TOPO) (RESPONDER O QUESTIONÁRIO)

- Coordenadores de cursos de graduação na área de computação e processamento de dados
- Coordenadores de cursos de graduação em quaisquer áreas, desde que o curso mantenha disciplinas de programação em seu currículo.

Os autores desta pesquisa:

(VOLTA AO TOPO) (RESPONDER O QUESTIONÁRIO)

Esta pesquisa é motivada por um projeto de pesquisa que está sendo desenvolvido no Curso de Pós-Graduação em Informática da UFPB pelas pessoas abaixo.

Orientador: Francisco Vilar Brasileiro, Phd

-
  - Professor adjunto da UFPB-DSC
  - Bacharel em Ciências da Computação
  - Doutor pela Universidade de Newcastle - Inglaterra
  - Áreas de interesse: Sistemas distribuídos, Tolerância a falhas, Educação em Informática

Mestrando: Dênio Mariz Timóteo

- Professor de Informática da Escola Técnica Federal da Paraíba
- Bacharel em Ciências da Computação
- Especialista em Informática Educativa - CEFET-MG
- Mestrando em informática pela UFPB-DSC
- Áreas de interesse: Educação em Informática, Bancos de dados, Sistemas operacionais

Mestrando: Severino do Ramo de Paiva

- Professor de Informática da Escola Técnica Federal da Paraíba
- Professor de Informática da Faculdade Paraibana de Processamento de Dados
- Bacharel em Ciências da Computação
- Mestrando em informática pela UFPB-DSC
- Áreas de interesse: Educação em Informática, Bancos de dados, Sistemas operacionais

---

1. Dados sobre o informante:

Nome completo (opcional):

E-mail (opcional):

Função atual:

Experiência na área de ensi

---

2. Dados sobre o curso:

Nome do curso em questão:

Bacharelado em Ciências da Computação

Nome do curso (caso não listad

Instituição de ensino:

Número médio de alunos formado

País:

---

3. Por favor forneça informações sobre as disciplinas de programação existentes no curso em questão.

- No campo *Pré-requisitos* informe ***o(s) nome(s)*** das disciplinas que devem ser cursadas antes
- No campo *Ementa resumida* informe o conteúdo da disciplina (texto livre).

Nome da Disciplina: / Pré-requisitos: / Carga horária semanal:	Ementa resumida:
Nome da Disciplina: / Pré-requisitos: / Carga horária semanal:	Ementa resumida:
Nome da Disciplina: / Pré-requisitos: / Carga horária semanal:	Ementa resumida:
Nome da Disciplina: / Pré-requisitos: / Carga horária semanal:	Ementa resumida:
Nome da Disciplina: / Pré-requisitos: / Carga horária semanal:	Ementa resumida:
Nome da Disciplina: / Pré-requisitos: / Carga horária semanal:	Ementa resumida:

4. Com respeito (apenas) ao ensino de programação, qual a sua opinião sobre as afirmações abaixo ?

	É necessário estabelecer uma separação entre o ensino de ***programação*** e o ensino de ***linguagens*** de programação para o alcance de melhores resultados.
	As disciplinas ligadas ao ensino de programação carecem de uma padronização de conteúdos nos cursos de graduação de seu conhecimento.
	A seqüência na qual os conteúdos das disciplinas ligadas ao ensino de programação são ministrados é importante para obtenção de melhores resultados.
	A seqüência na qual os conteúdos das disciplinas ligadas ao ensino de programação são ministrados varia nos cursos de graduação de seu conhecimento.
	É necessário que o ensino de programação seja menos dependente dos paradigmas de programação, para formar profissionais mais adaptáveis às evoluções que ocorrem na área.
	Depois de graduados, os profissionais no mercado conseguem se adaptar com facilidade às mudanças e evoluções que ocorrem na área de programação.
	A parte prática do ensino de programação é mais importante do que a parte teórica
	A parte prática e a parte teórica do ensino de programação são igualmente importantes
	As ferramentas de ensino auxiliado por computador (tutoriais de software) trazem benefícios ao aprendizado de programação, quando bem aplicadas

---

5. Você tem alguma sugestão ou comentário sobre esta pesquisa ou alguma experiência de sucesso a relatar quanto ao alcance de melhores resultados no ensino de programação ?

---

6. Autoriza citar ou fazer referências às informações sobre o curso ou experiências relatadas ?

- ○ Não, apenas contabilize as informações nas estatísticas
  ◉ Sim, pode citar ou fazer referências às informações aqui reveladas

---

Enviar Dados | Limpar Tela

Estas informações contribuirão para o desenvolvimento de um projeto de pesquisa sendo desenvolvido no Curso de Pós-Graduação em Informática da UFPB. Caso deseje fazer algum comentário adicional, favor contactar as pessoas abaixo.

- **Mestrando: Dênio Mariz - UFPB-DSC**
  Mestrando: Severino do Ramo de Paiva - UFPB-DSC
  Orientador: Francisco Vilar Brasileiro, Phd - UFPB-DSC

Muito obrigado pela sua contribuição !

Erro: Os dados não foram processados !

As seguintes informações são consideradas imprescindíveis para nossa pesquisa, mas não foram corretamente preenchidas:

Experiencia com ensino de programação
Universidade ou Instituição de ensino
Função que exerce atualmente
Número de alunos formados por ano (em média)

Por favor retorne ao formulário e preencha tais informações corretamente. Obrigado !

Os dados foram armazenados com sucesso !

Obrigado por ter participado da nossa pesquisa. Suas informações são muito importantes para nós.

Caso você tenha fornecido seu e-mail, teremos prazer em enviar-lhe os resultados desta pesquisa, no final do nosso trabalho.

Pedimos a gentileza de divulgar esta pesquisa com seus colegas, fornecendo-lhes o URL do questionário da página anterior.

Anexo B. Instituições participantes da Pesquisa Sobre o Ensino de Programação nas IES Brasileiras

#	Sigla	Instituição	Curso[9]	Contato	Email
1.	CRTSE	Centro Regional de Tecnologia Santa Escolástica	tpd	Roberto Gianolla	centro@objetivosorocaba.g12.br
2.	USP-SC	Escola de Engenharia de Sao Carlos – USP	eco	Jose Alberto Cuminato	jacumina@lcad.icmsc.sc.usp.br
3.	FPPD	Faculdade Paraibana de Processamento de Dados	tpd	Severino do Ramo de Paiva	paiva@dsc.ufpb.br
4.	SANTANNA	Faculdade Sant´Anna	bcc	Roberto Affonso da Costa Junior	affonso@comp.ita.cta.br
5.	TAPAJOS	Faculdade Tapajós	tpd	Edson Pinheiro Pimentel	eppsp@hotmai.com
6.	FESP	Fundação de Estudos Sociais do Paraná	tpd	Edson Martins Lecheta	lecheta@bsi.com.br
7.	IDEST	Instituto Diocesano de Ensino Sto.Antonio	tpd	Wladmir Janousek	janousek@aquarius.com.br
8.	PUCCAMP	Pontificia Universidade Catolica de Campinas – PUCCAMP	eco	Ivan Granja	ivan@zeus.puccamp.br
9.	PUCRJ	Pontificia Universidade Catolica do Rio de Janeiro	tpd	Roberto Lerusalimschy	roberto@inf.puc-rio.br
10.	PUCRS	Pontificia Universidade Catolica do Rio Grande do Sul – Uruguaiana	bin	Ney Laert Vilar Calazans	calazans@andros.inf.pucrs.b
11.	PUCRS	Pontificia Universidade Catolica do Rio Grande do Sul – Uruguaiana	bcc	Ney Vinas Lemos	nlemos@music.pucrs.br
12.	UNESP	Unesp – Campus Bauru	bcc	João Pedro Albino	jpalbino@azul.bauru.unesp.br
13.	UNESP	Unesp – Campus Rio Claro	bcc	João Pedro Albino	jpalbino@azul.bauru.unesp.br
14.	UNESP	Unesp – Campus São José do Rio Preto	bcc	João Pedro Albino	jpalbino@azul.bauru.unesp.br
15.	UNICAMP	Unicamp – Instituto de Computacao	bcc	Heloisa Vieira da Rocha	heloisa@dcc.unicamp.br
16.	UNICAP	Universidade Catolica de Pernambuco	bcc	Jairson Vitorino dos Santos Filho	jvsf@di.ufpe.br
17.	UCS	Universidade de Caxias do Sul	bcc	Cristian Koliver	ckoliver@lcmi.ufsc.br
18.	UEM	Universidade Estadual de Maringá	tpd	Airton Marco Polidoro	ampolido@din.uem.br
19.	UEM	Universidade Estadual de Maringá	bcc	Jose Roberto Vasconcelos	jrvasco@din.uem.br
20.	UEM	Universidade Estadual de Maringá	bin	Airton Marco Polidoro	ampolido@din.uem.br
21.	UNIOESTE	Universidade Estadual do Oeste do Paraná	ini	Moacir Gimenes Brito	mgbrito@unioeste.unioeste.br
22.	UFBA	Universidade Federal da Bahia	bcc	Jose Ulisses Ferreira Junior	ulisses@ufba.br
23.	UFPB	Universidade Federal da Paraíba – campus i	bcc	Gustavo Motta	gustavo@lasic.ufpb.br
24.	UFPB	Universidade Federal da Paraíba – campus ii	bcc	Jorge Cesar A. de Figueiredo	abrantes@dsc.ufpb.br
25.	UFAL	Universidade Federal de Alagoas	bcc	Olival de Gusmão Freitas	olival@dcc.ufal.br
26.	UFPE	Universidade Federal de Pernambuco	bcc	Eduardo Sodre	easodre@npd.ufpe.br
27.	UFPE	Universidade Federal de Pernambuco	ene	Jairo Simião Dornelas	jairo@npd.ufpe.br
28.	UFSC	Universidade Federal de Santa Catarina	bcc	Leandro J. Komosinski	leandro@inf.ufsc
29.	UFSCar	Universidade Federal de São Carlos	bcc	Celio Estevam Moron	dcem@power.ufscar.br
30.	UFSCar	Universidade Federal de São Carlos	eco	Lucia Helena Machado Rino	lucia@dc.ufscar.br
31.	UFU	Universidade Federal de Uberlândia	bcc	Sergio de Melo Schneider	sms@di.ufpe.br
32.	UFPR	Universidade Federal do Paraná	bin	Bruno Muller Junior	bruno@inf.ufpr.br
33.	UFRJ	Universidade Federal do Rio de Janeiro	bcc	Miguel Jonathan	jonathan@nce.ufrj.br
34.	UFRGS	Universidade Federal do Rio Grande do Sul	bcc	Daltro Jose Nunes	daltro@inf.ufrgs.br

Tabela 11 – Lista das Instituições de Ensino consultadas na pesquisa sobre ensino de programação

9 tpd = Tecnologia em Processamento de Dados; eco = Engenharia da Computação; bcc = Ciência da Computação; bin = Bacharelado em Informática; ini = Informática Industrial; ene = Engenharia Elétrica

Anexo C. Formulário da Pesquisa sobre a prática dos profissionais de programação

Nas páginas seguintes, o formulário eletrônico da Pesquisa 2 é apresentado na íntegra, de acordo como foi mostrado para o usuário através da *Web*. O fluxograma apresentado na Figura 31 abaixo descreve a seqüência de páginas apresentadas ao usuário, a partir de uma página básica comum às duas pesquisas. A figura mostra também as ações executadas pelo mecanismo de controle e recepção dos dados, que efetuava a consistência e armazenamento para posterior tratamento, o qual foi baseado na tecnologia CGI – *Commom Gateway Interface*.

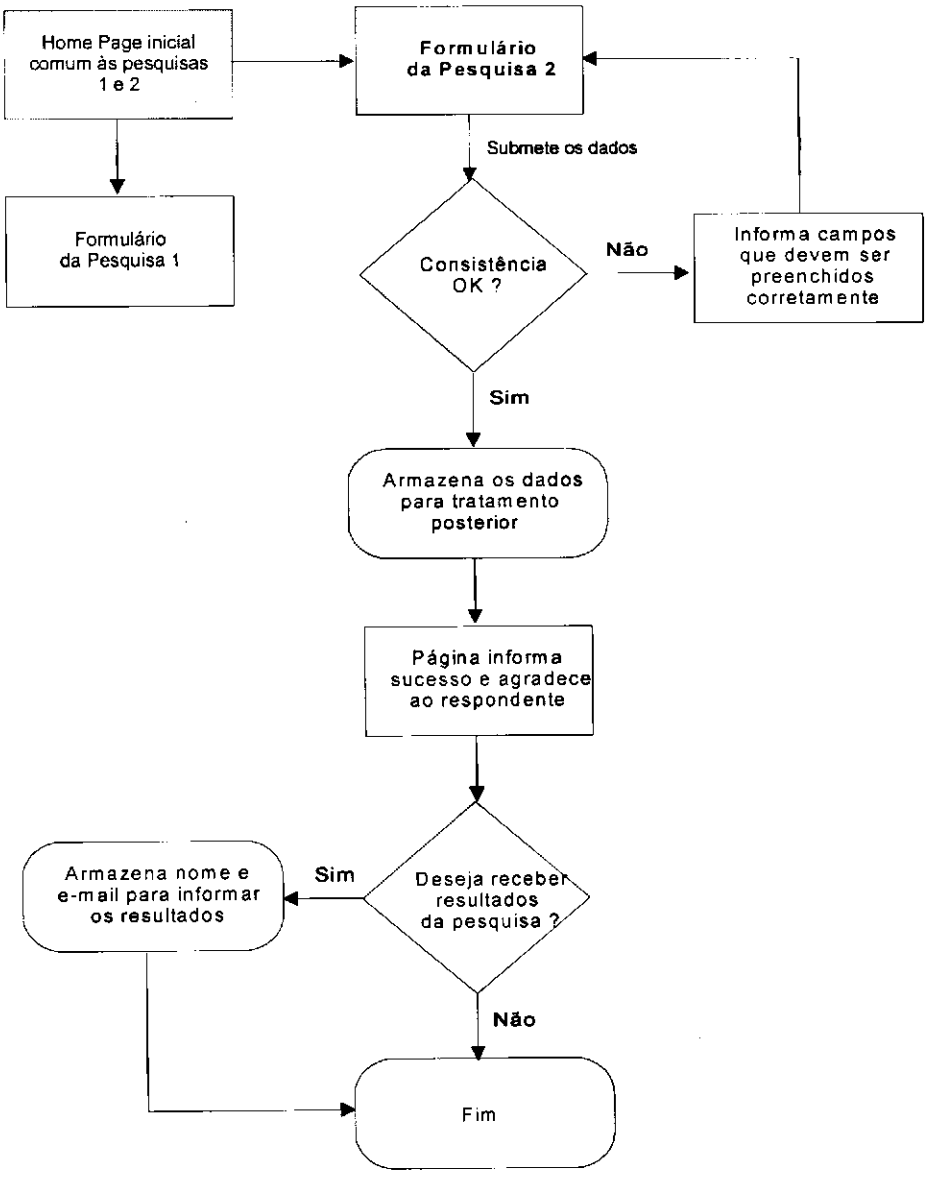

Figura 31 - Fluxo de informações do formulário da Pesquisa 2

UNIVERSIDADE FEDERAL DA PARAÍBA

Coleta de informações sobre a prática dos profissionais de programação

English

Sim, quero responder o questionário !

Objetivos:

(VOLTA AO TOPO) (RESPONDER O QUESTIONÁRIO)

Este questionário foi preparado para coletar dados para uma pesquisa desenvolvida por alunos de mestrado do Curso de Pós-Graduação em Informática da UFPB. Ao respondê-lo, você estará contribuindo com sua experiência para o estudo e desenvolvimento de uma nova metodologia para o ensino de programação. Se você informar seu e-mail no questionário, teremos o prazer de enviar-lhe os resultados da pesquisa.

Quem deve responder:

(VOLTA AO TOPO) (RESPONDER O QUESTIONÁRIO)

- Profissionais ligados à área de programação, com qualquer formação de nível superior.
- Programadores com formação superior
- Analistas de sistemas
- Gerentes de projetos de software

Os autores desta pesquisa:

(VOLTA AO TOPO) (RESPONDER O QUESTIONÁRIO)

Esta pesquisa é motivada por um projeto de pesquisa que está sendo desenvolvido no Curso de Pós-Graduação em Informática da UFPB pelas pessoas abaixo.

Orientador: Francisco Vilar Brasileiro, Phd

- Professor adjunto da UFPB-DSC
- Bacharel em Ciências da Computação
- Doutor pela Universidade de Newcastle - Inglaterra
- Áreas de interesse: Sistemas distribuídos, Tolerância a falhas, Educação em Informática

Mestrando: Dênio Mariz Timóteo

- Professor de Informática da Escola Técnica Federal da Paraíba
- Bacharel em Ciências da Computação
- Especialista em Informática Educativa - CEFET-MG
- Mestrando em informática pela UFPB-DSC
- Áreas de interesse: Educação em Informática, Bancos de dados, Sistemas operacionais

Mestrando: Severino do Ramo de Paiva

- Professor de Informática da Escola Técnica Federal da Paraíba
- Professor de Informática da Faculdade Paraibana de Processamento de Dados
- Bacharel em Ciências da Computação
- Mestrando em informática pela UFPB-DSC
- Áreas de interesse: Educação em Informática, Bancos de dados, Sistemas operacionais

---

Sim, quero responder o questionário !

UNIVERSIDADE FEDERAL DA PARAÍBA

Coleta de informações sobre a prática dos profissionais de programação

---

1. Dados sobre o informante:

Função que exerce atualmente:

Experiência com programação (em a

Nome do curso que concluiu: Bacharelado em Ciências da Computação

Nome do curso (caso não listado):

Universidade ou Instituição de ensino:

Ano da graduação:

País:

---

2. Selecione a ordem cronológica em que voce aprendeu as linguagens durante o seu curso de graduação. Caso a linguagem não esteja listada, favor digitar ao lado.

Primeiro lugar: Nenhuma — Outra:

Segundo lugar: Nenhuma — Outra:

Terceiro lugar: Nenhuma — Outra:

Quarto lugar: Nenhuma — Outra:

---

3. Selecione as linguagens nas quais voce apresenta maior domínio, independentemente da aplicação onde tal linguagm será usada. Caso a linguagem não esteja listada, favor digitar ao lado.

Primeiro lugar: Nenhuma — Outra:

Segundo lugar: Nenhuma — Outra:

Terceiro lugar: Nenhuma — Outra:

Quarto lugar: Nenhuma — Outra:

---

4. Desde quando se formou, que grau de dificuldade teve para se manter atualizado quanto às evoluções das técnicas de programação ?

○ Grande ○ Razoável ○ Pouca ○ Nenhuma

---

5. Com que freqüência você se utiliza dos recursos abaixo para se manter atualizado com as novas tecnologias de programação ?

	Cursos de atualização promovidos por universidades
	Cursos de atualização em congressos
	Treinamentos por conta própria
	Treinamentos externos, oferecidos pela empresa onde trabalho
	Treinamentos internos promovido pela empresa onde trabalho
	Seminários, palestras,workshops
	Peço orientação dos colegas profissionais
	Praticando, sem orientação formal
	Permaneço estudando o que já conheço bem, sem a preocupação de seguir tendências.
	Uso tutoriais disponíveis no mercado (software, video, hipertextos...)
	Outro (especifique):
	Outro (especifique):
	Outro (especifique):

---

6. Que grau de dificuldade você tem ao aprender uma NOVA linguagem ou se adaptar a uma INOVAÇÃO ligada à prática da programação ?

○ Grande ○ Razoável ○ Pouca ○ Nenhuma

6-a. Caso tenha respondido "grande" ou "razoável" à pergunta anterior, com que grau você acha que cada uma das opções abaixo contribuiu para este fato ? Por favor informe outras opções.

	À forma como aprendi a programar, muito vinculada a conceitos específicos da linguagem
	À forma como aprendi a programar, muito vinculada a conceitos da época
	Ao dimensionamento inadequado da prática da programação
	À inadequação da carga horária das disciplinas de programação
	Pouca dedicação individual às disciplinas de programação durante a graduação
	Superficialidade dos conteúdos das disciplinas de programação durante a graduação
	Outro (favor especifique):
	Outro (favor especifique):
	Outro (favor especifique):

6-b. Caso tenha respondido "pouca" ou "nenhuma" à pergunta 6, com que grau você acha que cada uma das opções abaixo contribuiu para este fato ? Por favor informe outras opções.

A forma como aprendi a programar, menos dependente de conceitos específicos da linguagem

A forma como aprendi a programar, muito independente de conceitos da época

O dimensionamento adequado da prática da programação nas disciplinas

A adequação da carga horária das disciplinas de programação

A minha experiência profissional alcançada na área de programação

A capacidade individual de adaptar-me a novos conceitos

Adequação dos conteúdos das disciplinas de programação durante a graduação

Outro (favor especifique):

Outro (favor especifique):

Outro (favor especifique):

Enviar Dados | Limpar Tela

Estas informações contribuirão para o desenvolvimento de um projeto de pesquisa sendo desenvolvido no Curso de Pós-Graduação em Informática da UFPB. Caso deseje fazer algum comentário adicional, favor contactar as pessoas abaixo.

Mestrando: Dênio Mariz - UFPB-DSC
Mestrando: Severino do Ramo de Paiva - UFPB-DSC
Orientador: Francisco Vilar Brasileiro, Phd - UFPB-DSC

Muito obrigado pela sua contribuição !

Erro: Os dados não foram processados !

As seguintes informações são consideradas imprescindíveis para nossa pesquisa, mas não foram corretamente preenchidas:

Função que exerce atualmente
Experiência com programação (em anos)
Universidade ou Instituição de ensino
Ano da Graduação
País

Por favor retorne ao formulário e preencha tais informações corretamente. Obrigado !

Os dados foram armazenados com sucesso !

Deseja receber os resultados desta pesquisa ?

Forneça seu nome e seu e-mail e teremos prazer em enviar-lhe os resultados desta pesquisa, no final do nosso trabalho. Para sua privacidade, seu nome não estará associado as suas respostas.

Nome Completo:	E-mail:

Envie-me os resultados da pesquisa

Obrigado pela sua participação !

Dê a oportunidade a seus colegas de se pronunciarem e enriquecerem esta pesquisa: divulgue o URL www.dsc.ufpb.br/~denio/p2.html.

Anexo D. Formulário da Pesquisa sobre a prática dos profissionais de programação – Versão em Inglês

Nas páginas seguintes, a versão em inglês do formulário eletrônico da Pesquisa 2 é apresentado na íntegra, de acordo como foi mostrado para o usuário através da *Web*. O fluxo de informações para este formulário é o mesmo da sua versão em português, apresentado no Anexo C. Os dados coletados por ambas as versões do formulário da Pesquisa 2 eram armazenados em um mesmo repositório de dados. Entretanto, um atributo adicional identificava qual a versão do formulário usado pelo respondente, para permitir uma correspondência aos interessados na língua correspondente, informando-lhes dos resultados da pesquisa.

FEDERAL UNIVERSITY OF PARAÍBA

Information Pool on Programming Practices

Yes, I want to answer the form !

Objectives:

(GO TO TOP) (ANSWER THE FORM)

This form was prepared to collect data for a research developed by M.Sc. students in Computing. When answering it, you are contributing with your experience for the study and development of a new methodology for teaching programming languages.

Who should answer it:

(GO TO TOP) (ANSWER THE FORM)

- Graduate professionals related to the programming area
- Graduate programmers
- Graduate system analysts
- Managers of software projects

The authors of this research:

(GO TO TOP) (ANSWER THE FORM)

This research project is being developed in the Computing Post-Graduate Course at Federal University of Paraíba (UFPB) by:

Adviser: Francisco Vilar Brasileiro, Ph.D.

- Senior lecturer in Computing at Federal University of Paraíba UFPB-DSC
- Bachelor in Computing Science
- Ph.D. by University of Newcastle Upon Tyne - England
- Interest areas: Distributed Systems, Fault Tolerance, Education in Computing

M.Sc. Student: Dênio Mariz Timóteo

- Lecturer in Computing at Federal Technical School of Paraíba
- Bachelor in Computing Science
- Specialist in Computing in Education - CEFET-MG
- M.Sc. student in Computing at Federal University of Paraíba - UFPB-DSC
- Interest areas: Education in Computing, Database Systems, Operating Systems

M.Sc. Student: Severino do Ramo de Paiva

- Lecturer in Computing at Federal Technical School of Paraíba
- Lecturer in Computing at Faculdade Paraibana de Processamento de Dados
- Bachelor in Computing Science
- M.Sc. student in Computing at Federal University of Paraíba - UFPB-DSC
- Interest areas: Education in Computing, Database Systems, Operating Systems

---

Yes, I want to answer the form !

FEDERAL UNIVERSITY OF PARAÍBA

Information Pool on Programming Practices

---

1. Personal data:

Current occupation or job:

Experience in programming (years)

Graduate course name: Computer Science

Graduate Course name (if not pres

Institution where have got the degree:

Year of completion (YYYY):

Country:

---

2. Fill in the chronological order in which you have learned the programming languages during your graduate course. If the language is not listed, please type in the space provided.

First place: None Other:

Second place: None Other:

Third place: None Other:

Fourth place: None Other:

---

3. Fill in the programming languages in which you have got more knowledge, independently of the application in which this language will be used. If the language is not listed, please type in the space provided.

First place: None Other:

Second place: None Other:

Third place: None Other:

Fourth place: None Other:

---

4. After you have graduate, which level of dificulty have you had to keep yourself up to date

with the evolutions of programming techniques?

○ A lot ○ Reasonable ○ Little ○ None

---

5. How often do you use the resources listed below to keep yourself up to date with new programming technologies?

	Updating courses offered by universities
	Updating courses offered in congresses/symposiums/etc.
	Self-training
	External training offered by the company where I work
	Internal training offered by the company where I work
	Seminars, talks, workshops
	Orientation from colleagues
	Practising without formal orientation
	Keep studying what I know well, without worrying about new trends
	Use tutorials available in the makret (software, video, hypertexts...)
	Other (please quantify and specify):
	Other (please quantify and specify):
	Other (please quantify and specify):

---

6. Which level of difficulty you generally have either to learn a *new* programming language or to adapt yourself to an *innovation* related to the programming practice?

○ A lot ○ Reasonable ○ Little ○ None

6-a. If you have answerd "A lot" or "Reasonable" to question 6, how much do you think each statement below have contributed for that? (Please quantify and inform other probable causes that are not listed)

The way I have learned how to program, much related to specific concepts of language

The way I have learned how to program, much related to specific concepts of that time

The inadequate duration of practice classes in programming disciplines

The inadequate number of hours to cover the contents in the programming disciplines

Little individual dedication to the programming subjects studied during the degree course

Superficial contents of the programming disciplines during the degree course

Other (please quantify and specify):

Other (please quantify and specify):

Other (please quantify and specify):

6-b. If you have answerd "Litle" or "None" to question 6, how much do you think each

statement below have contributed for that? (Please quantify and inform other probable causes that are not listed)

[ ] The way I have learned how to program, much independent of specific concepts of programming languages

[ ] The way I have learned how to program, much independent of concepts of that time

[ ] The adequate duration of practice classes in programming disciplines

[ ] The adequate number of hours to cover the contents in the programming disciplines

[ ] My personal experience reached in the programming area

[ ] The individual capacity to adapt to new concepts

[ ] The adjustment of the contents to the programming subjects studied during the degree course

[ ] Other (please quantify and specify): [ ]

[ ] Other (please quantify and specify): [ ]

[ ] Other (please quantify and specify): [ ]

Send Data | Clear

All this information will contribute to the development of a research project which is being developed in the Post-Graduate Course in Computing at Federal University of Paraíba. If you want to add any other comment, please contact the people below:

M.Sc. Student: Dênio Mariz - UFPB-DSC
M.Sc. Student: Severino do Ramo de Paiva - UFPB-DSC
Adviser: Francisco Vilar Brasileiro, Phd - UFPB-DSC

Thank you for your participation !

Error: The data were not processed !

The following informations are considered important to our research, but it were not filled in:

Current ocupation or job
Experience in programming (years)
Instituition were you have got the degree
Year of completion
Country

Please return to the form and fill in these informations. Thank you!

The data was saved successfuly !

Do you want to receive the results of this survey ?

Give us your name and e-mail and we will send you the results of this survey at the end of our research. For your privacy, your name will not to be associated with your aswers.

Full name:	E-mail:

Send me the results of this survey

Thanks for your participation !

Let your colleagues to know about this survey: divulge this URL
www.dsc.ufpb.br/~denio/e2.html

Anexo E. Resumo das características dos paradigmas de programação

O Paradigma de Programação Imperativo

Programação imperativa é assim chamada pelo fato de ser baseada em comandos que atualizam os valores de variáveis mantidas na memória (a palavra *imperativo* é derivada do Latim *imperativu* que significa *comando*). Este paradigma é relativamente antigo pelo fato dos projetistas da década de 1950 terem reconhecido que variáveis e comandos de atribuição constituíam uma abstração simples mas útil das instruções executadas pelo conjunto de instruções de máquina para operações com a memória. Devido ao seu próximo relacionamento com as arquiteturas de máquinas, as linguagens de programação imperativas podem, pelo menos a princípio, ser implementadas de forma muito eficiente.

O paradigma imperativo ainda é dominante hoje. A grande maioria dos softwares comerciais atualmente em uso ou em desenvolvimento é escrito em linguagem imperativa. A maioria dos programadores profissionais são peritos principalmente ou exclusivamente em linguagens de programação imperativas (pelo fato dos paradigmas *concorrente* e *orientado a objetos* serem subparadigmas imperativos, conforme discutiremos adiante, seus praticantes são também considerados programadores imperativos.)

Nós poderíamos argumentar que a dominação continuada do paradigma imperativo é devida inteiramente à inércia. Afinal, as linguagens dominantes são *Basic*, *Cobol* e *FORTRAN*, não as posteriores e melhores projetadas *Pascal* e *Ada*, portanto a inércia é certamente um fator. Contudo, existe uma razão mais fundamental para a importância das linguagens imperativas que é relacionada à natureza e à finalidade da programação. Programas são escritos para modelarem processos do mundo real que afetam objetos[10] do mundo real, os quais freqüentemente adquirem um estado que varia com o tempo. Variáveis modelam naturalmente tais objetos e programas imperativos modelam naturalmente tais processos. Portanto o uso de variáveis e dos comandos que produzem mudanças de estado sobre elas é um recurso muito apropriado para modelagens de processos, que compõem a maioria dos problemas abordados pela programação até agora.

Algumas Linguagens pertencem ao paradigma imperativo são: Pascal, C, COBOL, FORTRAN, BASIC, Ada, Assembly, Clipper, entre outras.

---

[10] A palavra "objeto" deve ser interpretada literalmente, sem vinculação ao conceito da palavra no contexo da programação orientada a objetos – POO.

O paradigma de Programação Concorrente

Segundo [Watt-91] seria um exagero reinvidicar que existe alguma coisa chamada "o" paradigma concorrente para uma disciplina que é ainda é relativamente imatura e para a qual várias abordagens estão sendo ainda perseguidas. Uma conseqüência infeliz disto é o fato de que programas concorrentes devam ser freqüentemente estruturados de acordo com uma particular arquitetura de hardware concorrente e não se adaptar bem a arquiteturas que suportam paradigmas dissimilares. Muitas pesquisas estão sendo dirigidas para a solução deste problema, o qual é talvez o principal obstáculo para o uso mais amplo de concorrência.

O uso de concorrência está presente em sistemas disponíveis para usuários finais, incluindo a maioria dos sistemas operacionais e aplicações dirigidas à simulação. Algumas linguagens de programação que representam o paradigma concorrente são: Ada, Linda, Occam, JPI Modula 2.

O paradigma de Programação Orientada a Objeto

Uma das deficiências da programação imperativa é que variáveis globais podem potencialmente serem acessadas (e atualizadas) por qualquer parte do programa. Programas grandes que fazem uso de alguma disciplina para acessar variáveis globais podem ficar ingerenciáveis. A razão é que nenhum módulo que acessa uma variável global pode ser desenvolvido e entendido independentemente dos outros módulos que também acessam aquela variável.

Este problema foi reconhecido por volta de 1970 por David Parnas, que pregava a disciplina da "ocultação da informação" como um remédio. Sua idéia era encapsular cada variável global em um módulo com um grupo de operações (tais como procedimentos e funções) que apenas tinham o direito de acessar a variável. Demais módulos podem acessar as variáveis apenas indiretamente, chamando essas operações. Nós agora usamos o termo *objeto* para designar esses módulos. Programação Orientada a Objetos é a disciplina que se baseia em objetos para impor uma estrutura modular aos programas.

Programação Orientada a Objetos é implementada de forma mais segura em linguagens que suportam o conceito de encapsulamento. Um objeto pode ser implementado por um "pacote" que exporta operações para manipularem de forma disciplinada as variáveis e mantém as variáveis escondidas.

Muitas linguagens imperativas que suportam encapsulamento, mantém a disciplina de orientação a objeto como uma opção a ser adotada ou não pelo programador. Por outro lado, uma linguagem *puramente* orientada a objetos remove esta opção e obriga que os

programas sejam construídos a partir de objetos. Este é o caso da linguagem *Smalltalk*. Outras linguagens de programação orientadas a objeto são: Simula-67, Eiffel, C++, Pascal OOP, Delphi (baseada em Pascal OOP).

O Paradigma de Programação Funcional

Nós freqüentemente podemos pensar em um programa como a implementação de um mapeamento. O programa obtém alguns valores de entrada e então os mapeia para valores de saída. Em programação imperativa, este mapeamento é alcançado indiretamente, por comandos que lêem valores de entrada, manipula-os e escreve valores de saída. Os comandos influenciam uns aos outros por meio de variáveis mantidas na memória e o relacionamento entre dois comandos pode ser completamente entendido apenas com referência a *todas* as variáveis que eles acessam e com referência a *todos* os outros comandos que acessam as mesmas variáveis. A menos que o programa seja escrito com cuidado e disciplina, estes relacionamentos entre comandos podem ser muito complicados e difíceis de entender.

Enquanto programação imperativa faz uso de variáveis comandos e procedures, a programação funcional é caracterizada pelo uso de expressões e funções. Em programação funcional o mapeamento de valores de entrada para valores de saída é alcançado mais diretamente. O programa *é* uma função (ou um grupo de funções) tipicamente compostas a partir de funções mais simples. Os relacionamentos entre funções são muito simples: uma função pode chamar outra, ou o resultado de uma função pode ser usado como argumento para outra função. Variáveis, comandos e efeitos colaterais são excluídos; ao invés disso, programas são escritos inteiramente dentro da linguagem de expressões, funções e declarações. Pode parecer estranho programar sem o uso de variáveis temporárias, mas em programação funcional a carência de armazenamento é compensada pela extensiva exploração de outros poderosos conceitos, como *funções de alta ordem* (higher-order functions) e *avaliação tardia* (lazzy evaluation).

Algumas linguagens que usam o paradigma funcional são: Lisp, Scheme, ML, Miranda.

O Paradigma de Programação Lógica

Todos os outros paradigmas anteriores são baseados na noção de que um programa implementa um mapeamento, sendo as saídas dependentes funcionalmente das entradas.

O paradigma *lógico* é baseado na noção de que um programa implementa uma relação, ao invés de um mapeamento de entrada-saída. Desde que as relações são mais gerais que mapeamentos, programação lógica é potencialmente de mais alto nível que as linguagens imperativas e funcionais.

Para ilustrar melhor o uso de uma linguagem lógica, considere dois conjuntos de valores *S* e *T* e uma relação *R* entre *S* e *T*, sendo para cada x ∈ S e y ∈ T, *R(x, y)* retorna ou TRUE ou FALSE. Por exemplo, '>' é uma relação entre números, uma vez que para cada par de números x e y, x>y é TRUE ou FALSE. (por convenção, escrevemos 'x>y' ao invés de '>(x,y)'). Outro exemplo simples é a relação "flui através" entre "rios" e "países": "o Amazonas flui através do Brasil" é TRUE, mas "O Nilo flui através da Austrália" é FALSE.

A principal representante do paradigma lógico é a linguagem Prolog.