

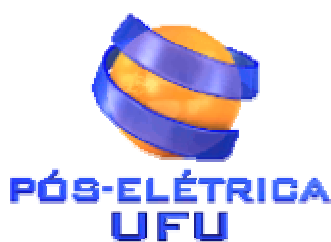


UNIVERSIDADE FEDERAL DE UBERLÂNDIA

TESE DE DOUTORADO

**MÉTODO TRIFÁSICO DE ENSINO-
APRENDIZAGEM BASEADO NA TAXIONOMIA DE
OBJETIVOS EDUCACIONAIS DE BLOOM: UMA
APLICAÇÃO NO ENSINO DE PROGRAMAÇÃO DE
COMPUTADORES**

EUSTÁQUIO SÃO JOSÉ DE FARIA



FACULDADE DE ENGENHARIA ELÉTRICA
Programa de Pós-graduação em Engenharia Elétrica



UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
Programa de Pós-Graduação em Engenharia Elétrica



Eustáquio São José de Faria

**MÉTODO TRIFÁSICO DE ENSINO-APRENDIZAGEM BASEADO
NA TAXIONOMIA DE OBJETIVOS EDUCACIONAIS DE BLOOM:
UMA APLICAÇÃO NO ENSINO DE PROGRAMAÇÃO DE
COMPUTADORES**

Tese apresentada à Faculdade de Engenharia Elétrica
da Universidade Federal de Uberlândia como parte
dos requisitos para a obtenção do título de Doutor
em Ciências.

Área de Concentração: Processamento da
Informação.

Orientador: Prof. Ph.D. Keiji Yamanaka

UBERLÂNDIA, 13 DE AGOSTO DE 2010.

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU – MG, Brasil

F224m Faria, Eustáquio São José de, 1973-

Método trifásico de ensino-aprendizagem baseado na taxionomia de objetivos educacionais de Bloom [manuscrito]: uma aplicação no ensino de programação de computadores / Eustáquio São José de Faria. - 2010. 295 f. : il.

Orientador: Keiji Yamanaka.

Tese (Doutorado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Engenharia Elétrica.

Inclui bibliografia.

1. Programação (Computadores) – Métodos de ensino – Teses. I. Yamanaka, Keiji. II. Universidade Federal de Uberlândia. Programa de Pós –Graduação em Engenharia Elétrica. III. Título.

CDU: **681.3.06: 371.3**



UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
Programa de Pós-Graduação em Engenharia Elétrica



Aluno: EUSTÁQUIO SÃO JOSÉ DE FARIA

Número da matrícula: 5061903

Área de concentração: PROCESSAMENTO DA INFORMAÇÃO

Título da dissertação:

MÉTODO TRIFÁSICO DE ENSINO-APRENDIZAGEM BASEADO NA TAXIONOMIA DE OBJETIVOS EDUCACIONAIS DE BLOOM: UMA APLICAÇÃO NO ENSINO DE PROGRAMAÇÃO DE COMPUTADORES.

Orientador: PROF. Ph.D. KEIJI YAMANAKA

A Tese foi **Aprovada** em sessão pública, realizada no Anfiteatro C do Bloco 50 da Universidade Federal de Uberlândia, em 13 de agosto de 2010, às 14:00 horas, com a seguinte Banca Examinadora:

Nome / Instituição:

Prof. Ph.D. Keiji Yamanaka / UFU

Prof. Dr. Alexsandro Santos Soares / UFGO

Prof. Dr. Celso Gonçalves Camilo Júnior / UFGD

Prof. Ph.D. Alexandre Cardoso / UFU

Prof. Dr. Luciano Vieira Lima / UFU

Prof. Keiji Yamanaka, Ph.D.
Orientador

Alexandre Cardoso, Ph. D.
Coordenador do Curso de Pós-Graduação

UBERLÂNDIA, 13 DE AGOSTO DE 2010.

*Ao Deus de todos os homens, à
Andréa Rosa pelo amor e
companheirismo, aos meus pais e
irmãos pelo incentivo e exemplo,
ao Gabrielzinho, por ter-nos
escolhido e ao Arthur e Luana,
por tê-los escolhido.*

AGRADECIMENTOS

Agradeço a todos os meus amigos (especialmente: Roberto Batata, Lulu Xan, Albert, Alex Mindu, Fábio e Jackson Clay) pelo incentivo durante outros períodos de minha vida acadêmica.

Aos meus companheiros da Faculdade de Engenharia Elétrica (especialmente: Eliane Elias, Aleandra, Celso, Alexandre Grings, Bruno, Juscélio e Luiz Fernando), que contribuíram de forma direta e indireta para a realização deste trabalho.

Aos meus companheiros de trabalho na PUC Minas (especialmente: Hermes Moraes, Marco Rodrigo, Alisson Arantes, Sandro Laudares, Fábio Martins, Vander Aguiar, Ramon Leite, Aluisio Eustáquio, Nilma, Jorge Sündermman, Ricardo Siervi, Laura e Cássio Castro).

Ao meu amigo e cardiologista Dr. Eduardo Augusto Victor Rocha.

Ao meu orientador, Keiji Yamanaka, pelas idéias e confiança no desenvolvimento da Tese.

À Universidade Federal de Uberlândia (Faculdade de Engenharia Elétrica) e à Pontifícia Universidade Católica de Minas Gerais, que forneceram o apoio necessário à realização da pesquisa.

FARIA, Eustáquio São José de *Método trifásico de ensino-aprendizagem baseado na taxionomia de objetivos educacionais de Bloom: uma aplicação no ensino de programação de computadores.* Tese de Doutorado, Faculdade de Engenharia Elétrica, Universidade Federal de Uberlândia, 2010. 295p.

RESUMO

O ensino-aprendizagem de programação de computadores tem sido alvo de investigações científicas desde os primórdios da computação. Embora diversas estratégias foram e ainda têm sido experimentadas, sua aplicação em cursos de computação e afins se mostra difícil, uma vez que exige algum empenho de educadores em buscá-las, estudá-las e avaliá-las. Não procurando a cura milagrosa para todos os problemas nessa área (relatados na literatura), e acreditando no potencial pedagógico da teoria do conflito sócio-cognitivo, decidiu-se construir um método de ensino de programação baseado na taxionomia de objetivos educacionais de Bloom a ser aplicado através da técnica de Programação em Duplas. O método está sendo utilizado na produção de um livro de Algoritmos e Técnicas de Programação no intuito de difundi-lo na Academia. Foram realizados experimentos com o método em uma Universidade da iniciativa privada e constatou-se um expressivo aumento de motivação dos aprendizes que a ele foram submetidos.

Palavras chave: Ensino-Aprendizagem de programação, Método de ensino, Programação em Duplas, Taxionomia de objetivos educacionais de Bloom, Teoria do conflito sócio-cognitivo.

FARIA, Eustáquio São José de *Three-phase method for teaching-learning technique based on Bloom's taxonomy of educational objectives: applied on computer-programming teaching*. Dr Thesis, College of Electric Engineering, Federal University of Uberlândia, 2010. 295p.

ABSTRACT

The Teaching and learning computer-programming process as been the target of scientific research since the early days of computing. Although several strategies have been and are still being tested, its application in computer-based courses has been difficult, since it requires some commitment from educators to search, study and evaluate them. Although not looking for a miraculous cure for all problems in this area (reported in the literature), and believing in the pedagogical potential of the socio-cognitive conflict theory, it has been decided to develop a teaching method of programming based on Bloom's taxonomy of educational objectives to be applied using the pair-programming technique. This method is being used to produce a book of Algorithms and Programming Techniques intending to be acknowledged by the Academy. Experiments with such method has been carried out at a private University and there has been a significant increase of motivation by the students that underwent the experiments.

Keywords: Programming Teaching-Learning, Teaching Method, Pair Programming, Bloom's Taxonomy of Educational Objectives, Socio-Cognitive Conflict Theory.

LISTA DE FIGURAS

Figura 2.1 – Avançando nas Categorias do Conhecimento Segundo o Domínio Cognitivo de Bloom.	33
Figura 2.2 – Comportamento Cognitivo para Atingir Alvos Afetivos.	58
Figura 2.3 – Comportamento Afetivo para Atingir Alvos Cognitivos.	59
Figura 4.1 – Faixas de Aprovação e Esquema de Nivelamento de Acordo com as Categorias do Domínio Cognitivo de Bloom.	83
Figura 5.1 – Fases do Método Trifásico de Ensino-Aprendizagem.	106
Figura 5.2 – Adquirindo Conhecimento de Conteúdos em Forma de Espiral.....	107
Figura 6.1 – Usando a Programação em Duplas Para atingir os Objetivos Educacionais nos Domínios Afetivo e Cognitivo.	112
Figura 7.1. Avaliação da Programação em Duplas.	155
Figura 7.2. Influência da PP no Aprendizado de Programação.....	155

LISTA DE TABELAS

Tabela 2.1 – Objetivos da Taxionomia de Bloom e Processos para Atingi-los – Domínio Cognitivo	63
Tabela 2.2 – Objetivos da Taxionomia de Bloom e Processos para Atingi-los – Domínio Afetivo	64
Tabela 4.1 – Distinguindo PP na Indústria de Software de PP na Academia.....	72
Tabela 5.1 – <i>Checklist</i> para Testar o MTEA na Fase 1 (de Acordo com Processos e Verbos de Ação)	109
Tabela 5.2 – <i>Checklist</i> para Testar o MTEA na Fase 2 (de Acordo com Processos e Verbos de Ação)	109
Tabela 5.3 – <i>Checklist</i> para Testar o MTEA na Fase 3 (de Acordo com Processos e Verbos de Ação)	110
Tabela 6.1 – Conteúdo Programático da Disciplina “Programação de Computadores”	113
Tabela 6.2 – Objetivos e Habilidades da Fase “Vejo, Escuto e Compreendo” para o Objetivo Específico “Compreender os Sistemas de Numeração e as Transformações de Base”	115
Tabela 6.3 – Objetivos e Habilidades da Fase “Discuto, Aplico e Reflito” para o Objetivo Específico “Compreender os Sistemas de Numeração e as Transformações de Base”	116
Tabela 6.4 – Objetivos e Habilidades da Fase “Pratico, Faço e Crio” para o Objetivo Específico “Compreender os Sistemas de Numeração e as Transformações de Base”	117
Tabela 6.5 – Objetivos e Habilidades da Fase “Vejo, Escuto e Compreendo” para o Objetivo Específico “Introduzir Algoritmos”	120
Tabela 6.6 – Objetivos e Habilidades da Fase “Discuto, Aplico e Reflito” para o Objetivo Específico “Introduzir Algoritmos”	121

Tabela 6.7 – Objetivos e Habilidades da Fase “Pratico, Faço e Crio” para o Objetivo Específico “Introduzir Algoritmos”	122
Tabela 6.8 – Objetivos e Habilidades da Fase “Vejo, Escuto e Compreendo” para o Objetivo Específico “Construir Expressões e Entender a Prioridade de Operadores”	124
Tabela 6.9 – Objetivos e Habilidades da Fase “Discuto, Aplico e Reflito” para o Objetivo Específico “Construir Expressões e Entender a Prioridade de Operadores”	125
Tabela 6.10 – Objetivos e Habilidades da Fase “Pratico, Faço e Crio” para o Objetivo Específico “Construir Expressões e Entender a Prioridade de Operadores”	126
Tabela 6.11 – Objetivos e Habilidades da Fase “Vejo, Escuto e Compreendo” para o Objetivo Específico “Definir Dados”	127
Tabela 6.12 – Objetivos e Habilidades da Fase “Discuto, Aplico e Reflito” para o Objetivo Específico “Definir Dados”	128
Tabela 6.13 – Objetivos e Habilidades da Fase “Pratico, Faço e Crio” para o Objetivo Específico “Definir Dados”	129
Tabela 6.14 – Objetivos e Habilidades da Fase “Vejo, Escuto e Compreendo” para o Objetivo Específico “Construir Tipos Abstratos de Dados”	131
Tabela 6.15 – Objetivos e Habilidades da Fase “Discuto, Aplico e Reflito” para o Objetivo Específico “Construir Tipos Abstratos de Dados”	132
Tabela 6.16 – Objetivos e Habilidades da Fase “Pratico, Faço e Crio” para o Objetivo Específico “Construir Tipos Abstratos de Dados”	133
Tabela 6.17 – Objetivos e Habilidades da Fase “Vejo, Escuto e Compreendo” para o Objetivo Específico “Conhecer os Tipos de Estruturas de Controle e Entender as Estruturas de Sequência”	135
Tabela 6.18 – Objetivos e Habilidades da Fase “Discuto, Aplico e Reflito” para o Objetivo Específico “Conhecer os Tipos de Estruturas de Controle e Entender as Estruturas de Sequência”	135
Tabela 6.19 – Objetivos e Habilidades da Fase “Pratico, Faço e Crio” para o Objetivo Específico “Conhecer os Tipos de Estruturas de Controle e Entender as Estruturas de Sequência”	136

Tabela 6.20 – Objetivos e Habilidades da Fase “Vejo, Escuto e Compreendo” para o Objetivo Específico “Entender as Estruturas de Seleção (Alternativas)”	137
Tabela 6.21 – Objetivos e Habilidades da Fase “Discuto, Aplico e Reflito” para o Objetivo Específico “Entender as Estruturas de Seleção (Alternativas)”	137
Tabela 6.22 – Objetivos e Habilidades da Fase “Pratico, Faço e Crio” para o Objetivo Específico “Entender as Estruturas de Seleção (Alternativas)”	138
Tabela 6.23 – Objetivos e Habilidades da Fase “Vejo, Escuto e Compreendo” para o Objetivo Específico “Entender as Estruturas de Repetição (Laços)”	139
Tabela 6.24 – Objetivos e Habilidades da Fase “Discuto, Aplico e Reflito” para o Objetivo Específico “Entender as Estruturas de Repetição (Laços)”	140
Tabela 6.25 – Objetivos e Habilidades da Fase “Pratico, Faço e Crio” para o Objetivo Específico “Entender as Estruturas de Repetição (Laços)”	142
Tabela 6.26 – Objetivos e Habilidades da Fase “Vejo, Escuto e Compreendo” para o Objetivo Específico “Entender Modularização e Construir Software Modular”	144
Tabela 6.27 – Objetivos e Habilidades da Fase “Discuto, Aplico e Reflito” para o Objetivo Específico “Entender Modularização e Construir Software Modular”	144
Tabela 6.28 – Objetivos e Habilidades da Fase “Pratico, Faço e Crio” para o Objetivo Específico “Entender Modularização e Construir Software Modular”	145
Tabela 7.1. Histórico dos dados dos alunos na disciplina de ATP	151
Tabela 7.2. Resultados dos Testes de Hipótese	153
Tabela 7.3. Índice de Satisfação dos Alunos com Programação	158

LISTA DE ABREVIATURAS

ATP	Algoritmos e Técnicas de Programação (Disciplina)
MTEA	Método Trifásico de Ensino-Aprendizagem
PC	Programação de Computadores (Disciplina)
PP	<i>Pair Programming</i>
SBIE	Simpósio Brasileiro de Informática na Educação
Zcal	Variável Z calculada através de teste de hipótese (Teste dos Sinais)

SUMÁRIO

INTRODUÇÃO.....	22
1.1 OBJETIVO GERAL.....	24
1.2 OBJETIVOS ESPECÍFICOS	24
1.3 PUBLICAÇÕES ORIGINADAS DESTE TRABALHO.....	25
1.4 MOTIVAÇÃO E JUSTIFICATIVA	26
1.5 ESTRUTURA DA TESE	27
CAPÍTULO 2	29
TAXIONOMIA DE OBJETIVOS EDUCACIONAIS.....	29
2.1 HISTÓRIA DA CRIAÇÃO DA TAXIONOMIA.....	30
2.2 O DOMÍNIO COGNITIVO	32
2.2.1 Conhecimento.....	33
2.2.1.1 Conhecimento de Específicos.....	34
2.2.1.1.1. Conhecimento de Terminologia	34
2.2.1.1.2. Conhecimento de Fatos Específicos	35
2.2.1.2 Conhecimento de Maneiras e Meios de Tratar com Específicos	35
2.2.1.2.1. Conhecimento de Convenções	35
2.2.1.2.2. Conhecimento de Tendências e Sequências	36
2.2.1.2.3. Conhecimento de Classificações e Categorias	36
2.2.1.2.4. Conhecimento de Critérios.....	37
2.2.1.2.5. Conhecimento de Metodologia	37
2.2.1.3 Conhecimento dos Universais e Abstrações em um Certo Campo	37
2.2.1.3.1. Conhecimento de Princípios e Generalizações.....	38
2.2.1.3.2. Conhecimento de Teorias e Estruturas.....	38
2.2.2 Capacidades e Habilidades Intelectuais.....	39
2.2.2.1 Compreensão	39
2.2.2.1.1. Tradução	39
2.2.2.1.2. Interpretação.....	40
2.2.2.1.3. Extrapolação.....	40
2.2.2.2 Aplicação.....	41
2.2.2.3 Análise.....	41

2.2.2.3.1. Análise de Elementos	42
2.2.2.3.2. Análise de Relações	42
2.2.2.3.3. Análise dos Princípios Organizacionais	42
2.2.2.4 Síntese	43
2.2.2.4.1. Produção de uma Comunicação Singular	43
2.2.2.4.2. Produção de um Plano ou Indicação de um Conjunto de Operações	44
2.2.2.4.3. Derivação de um Conjunto de Relações Abstratas	44
2.2.2.5 Avaliação	44
2.2.2.5.1. Julgamentos em Termos de Evidência Interna	45
2.2.2.5.2. Julgamentos em Termos de Critérios Externos	45
2.3 O DOMÍNIO AFETIVO	45
2.3.1 Acolhimento (atenção)	47
2.3.1.1 Percepção	48
2.3.1.2 Disposição para Receber	49
2.3.1.3 Atenção Controlada ou Seletiva	49
2.3.2 Resposta	50
2.3.2.1 Aquiescência na Resposta	50
2.3.2.2 Disposição para Responder	51
2.3.2.3 Satisfação na Resposta	51
2.3.3 Valorização	52
2.3.3.1 Aceitação de um Valor	52
2.3.3.2 Preferência por um Valor	53
2.3.3.3 Cometimento	53
2.3.4 Organização	54
2.3.4.1 Conceitualização de um Valor	54
2.3.4.2 Organização de um Sistema de Valores	55
2.3.5 Caracterização por um Valor ou Complexo de Valores	55
2.3.5.1 Direção Generalizada	56
2.3.5.2 Caracterização	56
2.4 O RELACIONAMENTO ENTRE OS DOMÍNIOS COGNITIVO E AFETIVO	57
2.4.1 Objetivos Cognitivos como Meio para Metas Afetivas	57
2.4.2 Objetivos Afetivos como Meio para Metas Cognitivas	58
2.4.3 Concluindo Sobre a Teoria do Relacionamento Entre os Dois Domínios	60
2.5 A TAXIONOMIA DE BLOOM REVISADA POR ANDERSON E COLABORADORES	61
2.5.1 A Dimensão do Conhecimento na Taxionomia Revisada	62
2.6 A TAXIONOMIA DE BLOOM CONDENSADA	63
2.7 CONCLUINDO SOBRE A TAXIONOMIA DE OBJETIVOS EDUCACIONAIS DE BLOOM	65

CAPÍTULO 3	66
PROGRAMAÇÃO EM DUPLAS (<i>PAIR PROGRAMMING</i> – PP).....	66
3.1 TEMAS CHAVE EM PROGRAMAÇÃO EM DUPLAS	69
3.1.1 Custo/Benefício de PP	70
3.1.2 PP na Indústria de Software Versus PP na Academia	72
3.1.3 Metodologias Para o Uso de PP no Ensino de Programação e na Indústria de Software.....	73
3.1.3.1 Rodízio dos pares	74
3.1.3.2 Troca de papéis.....	74
3.1.3.3 Escolha dos pares	75
3.1.4 Diretrizes Para o Bom Funcionamento de PP	75
3.1.5 Heterogeneidade Versus Homogeneidade em PP	76
3.1.6 O impacto de PP aplicado a Grupos Minoritários e às Mulheres.....	77
3.1.7 A Pressão dos Pares Decorrente de PP.....	78
3.2 POR QUÊ USAR PP NO ENSINO DE PROGRAMAÇÃO DE COMPUTADORES?.....	79
3.3 CONSIDERAÇÕES FINAIS IMPORTANTES PARA UMA IMPLANTAÇÃO BEM SUCEDIDA DE PP	79
CAPÍTULO 4	81
TRABALHOS RELACIONADOS	81
4.1 ESQUEMA DE NIVELAMENTO DE ESTUDANTES BASEADO EM CRITÉRIOS E NA TAXIONOMIA DE BLOOM.....	81
4.1.1 A Taxionomia de Bloom Aplicada aos Alunos de Acordo com Três Faixas de Aprovação.....	82
4.1.2 Concluindo Sobre o Esquema de Nivelamento de Estudantes de Lister e Leaney	83
4.2 A TAXIONOMIA DE TIPOS DE TAREFAS.....	84
4.2.1 As Tarefas.....	84
4.2.1.1 Tarefas Declarativas	85
4.2.1.2 Tarefas de Compreensão	85
4.2.1.3 Tarefas de Depuração	86
4.2.1.4 Tarefas de Predição	86
4.2.1.5 Tarefas de Provimento de Exemplo.....	87
4.2.1.6 Tarefas de Provimento de Modelo.....	87
4.2.1.7 Tarefas de Avaliação	88
4.2.1.8 Tarefas de Projeto.....	89
4.2.1.9 Tarefas de Resolução de Problemas	89

4.2.1.10 Tarefas de reflexão	90
4.2.2 Concluindo Sobre a Taxionomia de Tarefas de Bower	90
4.3 A REVISADA TAXIONOMIA DE BLOOM APLICADA PARA MELHORAR O PROCESSO DE AVALIAÇÃO DOS ALUNOS EM PROGRAMAÇÃO	91
4.3.1 As Categorias da Revisada Taxionomia de Bloom e Exemplos de Tarefas Avaliativas em Programação	92
4.3.1.1 Lembrar	92
4.3.1.2 Entender	93
4.3.1.3 Aplicar	94
4.3.1.4 Analisar	96
4.3.1.5 Avaliar	97
4.3.1.6 Criar	98
4.3.2 Concluindo Sobre a Taxionomia de Bloom Aplicada para Melhorar o Processo de Avaliação dos Alunos em Programação	99
4.4 OUTROS TRABALHOS NA ÁREA	100
4.5 CONCLUINDO SOBRE OS TRABALHOS RELACIONADOS ENCONTRADOS	101
CAPÍTULO 5	102
MÉTODO TRIFÁSICO DE ENSINO-APRENDIZAGEM (MTEA)	102
5.1 POR QUE USAR O DOMÍNIO COGNITIVO ?	102
5.2 POR QUE USAR O DOMÍNIO AFETIVO ?	104
5.3 DOMÍNIO COGNITIVO VERSUS AFETIVO	104
5.4 MÉTODO TRIFÁSICO DE ENSINO-APRENDIZAGEM	105
5.5 <i>CHECKLIST</i> PARA TESTAR O MTEA	107
5.6 OBSERVAÇÕES IMPORTANTES	110
CAPÍTULO 6	112
APLICANDO O MTEA NA DISCIPLINA “PROGRAMAÇÃO DE COMPUTADORES”	112
6.1 O MTEA NA DISCIPLINA “PROGRAMAÇÃO DE COMPUTADORES”	113
6.1.1 Compreender os Sistemas de Numeração e as Transformações de Base	114
6.1.2 Introduzir Algoritmos	118
6.1.3 Construir Expressões e Entender a Prioridade de Operadores	123
6.1.4 Definir dados	126
6.1.5 Construir Tipos Abstratos de Dados	130
6.1.6 Conhecer os Tipos de Estruturas de Controle e Entender as Estruturas de Sequência	134

6.1.7 Entender as Estruturas de Seleção (Alternativas).....	136
6.1.8 Entender as Estruturas de Repetição (Laços)	139
6.1.9 Entender Modularização e Construir Software Modular.....	142
6.2 OBSERVAÇÕES E DICAS IMPORTANTES.....	146
CAPÍTULO 7	148
RESULTADOS OBTIDOS NOS EXPERIMENTOS COM O MTEA	148
7.1 PRIMEIRA FASE DE EXPERIMENTOS	149
7.1.1 Primeiro Enfoque.....	149
7.1.1.1 Resultados do Primeiro Enfoque	151
7.1.1.2 Discutindo os Resultados do Primeiro Enfoque	154
7.1.2 Segundo Enfoque.....	154
7.1.2.1 Resultados do Segundo Enfoque	154
7.1.2.2 Discutindo os Resultados do Segundo Enfoque	156
7.1.3 Terceiro Enfoque	157
7.1.3.1 Resultados do Terceiro Enfoque.....	157
7.1.3.2 Discutindo os Resultados do Terceiro Enfoque.....	158
7.1.4 Problemas Encontrados na Primeira Fase de Experimentos	159
7.2 SEGUNDA FASE DE EXPERIMENTOS	160
7.2.1 Comportamentos Percebidos nas Seções de Emparelhamento Durante a Segunda Fase de Experimentos.....	161
7.2.1.1 Pressão da dupla	162
7.2.1.2 Conflito sócio-cognitivo.....	164
7.2.1.3 Aumento de coragem devido ao emparelhamento.....	165
7.2.1.4 Revisão contínua	165
7.2.1.5 Depuração em dupla.....	166
7.2.1.6 Aprendizagem em dupla.....	167
7.2.1.7 Confiança na parceria.....	168
7.2.2 Problemas Percebidos nas Seções de Emparelhamento Durante a Segunda Fase de Experimentos.....	169
7.2.2.1 A dependência	169
7.2.2.2 A falta da rotatividade	169
7.2.2.3 A obsessão pelo mais experiente	170
7.2.2.4 A dificuldade de criar seções de emparelhamento extraclasse	170
7.2.2.5 As discordâncias.....	171
7.2.2.6 O desequilíbrio de habilidades	171
7.2.2.7 A personalidade introvertida e a personalidade egocêntrica.....	172
7.2.2.8 A dominância/complacência.....	172
7.3 ALGUMAS OBSERVAÇÕES PARTICULARES.....	173

CAPÍTULO 8	174
CONCLUSÕES E TRABALHOS FUTUROS	174
REFERÊNCIAS	180
APÊNDICE A	191
EXERCÍCIOS PROPOSTOS PARA APLICAÇÃO DURANTE AS FASES DO MTEA ...	191
A.1. COMPREENDER OS SISTEMAS DE NUMERAÇÃO E AS TRANSFORMAÇÕES DE BASE	191
A.2. INTRODUZIR ALGORITMOS	195
A.3. CONSTRUIR EXPRESSÕES E ENTENDER A PRIORIDADE DE OPERADORES	200
A.4. DEFINIR DADOS.....	206
A.5. CONSTRUIR TIPOS ABSTRATOS DE DADOS	212
A.6. CONHECER OS TIPOS DE ESTRUTURAS DE CONTROLE E ENTENDER AS ESTRUTURAS DE SEQUÊNCIA	220
A.7. ENTENDER AS ESTRUTURAS DE SELEÇÃO (ALTERNATIVAS).....	230
A.8. ENTENDER AS ESTRUTURAS DE REPETIÇÃO (LAÇOS)	241
A.9. ENTENDER MODULARIZAÇÃO E CONSTRUIR SOFTWARE MODULAR	252
APÊNDICE B.....	261
EXERCÍCIOS DE LÓGICA	261
B.1. EXERCÍCIOS PARA TREINAR A ATENÇÃO.....	261
B.2. EXERCÍCIOS PARA TREINAR O RACIOCÍNIO LÓGICO	263
APÊNDICE C.....	268
COMO AGIR DURANTE O EMPARELHAMENTO DE PROGRAMADORES?.....	268
C.1 DIRETRIZ 1 – DETALHAMENTO DOS PASSOS E REGRAS DE PP	269
C.1.1 Instrutores ou gerentes de projeto devem	269
C.1.2 Estudantes ou desenvolvedores devem.....	270
C.2 DIRETRIZ 2 – COMUNICAÇÃO ATIVA ENTRE NAVEGADOR E PILOTO.....	271
C.2.1 Instrutores ou gerentes de projeto devem	271
C.2.2 Estudantes ou desenvolvedores devem.....	271

C.3 DIRETRIZ 3 – CULTURA ORIENTADA PARA A PP E PARA ATIVIDADES COLABORATIVAS	272
C.3.1 Instrutores ou gerentes de projeto devem	272
C.3.2 Estudantes ou desenvolvedores devem.....	273
C.4 DIRETRIZ 4 – INCENTIVO AO RELATO DE CASOS DE INCOMPATIBILIDADE	274
C.4.1 Instrutores ou gerentes de projeto devem	275
C.4.2 Estudantes ou desenvolvedores devem.....	275
C.5 DIRETRIZ 5 – HONESTIDADE, COLABORAÇÃO E SENSO DE RESPONSABILIDADE.....	275
C.5.1 Instrutores ou gerentes de projeto devem	276
C.5.2 Estudantes ou desenvolvedores devem.....	276
C.6 DIRETRIZ 6 – COMPATIBILIDADE DAS DUPLAS.....	277
C.6.1 Instrutores ou gerentes de projeto devem	277
C.6.2 Estudantes ou desenvolvedores devem.....	278
C.7 DIRETRIZ 7 – TRATAMENTO ESPECIAL AOS RESISTENTES	278
C.7.1 Instrutores ou gerentes de projeto devem	278
C.7.2 Estudantes ou desenvolvedores devem.....	279
C.8 DIRETRIZ 8 – EMPARELHAMENTOS OBRIGATÓRIOS EM LABORATÓRIOS	279
C.8.1 Instrutores ou gerentes de projeto devem	279
C.8.2 Estudantes ou desenvolvedores devem.....	279
C.9 DIRETRIZ 9 – COMPRIMENTO ESPERADO DAS TAREFAS	280
C.9.1 Instrutores ou gerentes de projeto devem	280
C.9.2 Estudantes ou desenvolvedores devem.....	280
C.10 DIRETRIZ 10 – PADRÃO DE CODIFICAÇÃO	281
C.10.1 Instrutores ou gerentes de projeto devem	281
C.10.2 Estudantes ou desenvolvedores devem.....	282
C.11 DIRETRIZ 11 – TROCA DE CONHECIMENTO ENTRE SÓCIOS	282
C.11.1 Instrutores ou gerentes de projeto devem	282
C.11.2 Estudantes ou desenvolvedores devem.....	282
C.12 DIRETRIZ 12 – SUCESSIVOS EMPARELHAMENTOS ALEATÓRIOS	283
C.12.1 Instrutores ou gerentes de projeto devem	283
C.12.2 Estudantes ou desenvolvedores devem.....	283

C.13 DIRETRIZ 13 – LIBERDADE DA TROCA DE PAPÉIS.....	284
C.13.1 Instrutores ou gerentes de projeto devem	284
C.13.2 Estudantes ou desenvolvedores devem.....	284
C.14 DIRETRIZ 14 – MOTIVAÇÃO DOS SÓCIOS.....	285
C.14.1 Instrutores ou gerentes de projeto devem	285
C.14.2 Estudantes ou desenvolvedores devem.....	285
C.15 DIRETRIZ 15 – FEEDBACK DE COMPORTAMENTO	286
C.15.1 Instrutores ou gerentes de projeto devem	286
C.15.2 Estudantes ou desenvolvedores devem.....	286
C.16 DIRETRIZ 16 – RODÍZIO DOS PARES	286
C.16.1 Instrutores ou gerentes de projeto devem	287
C.16.2 Estudantes ou desenvolvedores devem.....	287
C.17 DIRETRIZ 17 – PENSAMENTOS POSITIVOS.....	288
C.17.1 Instrutores ou gerentes de projeto devem	288
C.17.2 Estudantes ou desenvolvedores devem.....	288
C.18 DIRETRIZ 18 – EXTINÇÃO DO EGOCENTRISMO.....	289
C.18.1 Instrutores ou gerentes de projeto devem	289
C.18.2 Estudantes ou desenvolvedores devem.....	289
C.19 DIRETRIZ 19 – INCENTIVO À HUMILDADE	289
C.19.1 Instrutores ou gerentes de projeto devem	290
C.19.2 Estudantes ou desenvolvedores devem.....	290
C.20 DIRETRIZ 20 – PERÍODO DE TEMPO DEDICADO À PAIR PROGRAMMING	290
C.20.1 Instrutores ou gerentes de projeto devem	291
C.20.2 Estudantes ou desenvolvedores devem.....	291
C.21 DIRETRIZ 21 – EMPARELHAMENTOS EM PROJETOS	291
C.21.1 Instrutores ou gerentes de projeto devem	292
C.21.2 Estudantes ou desenvolvedores devem.....	292
C.22 DIRETRIZ 22 – DEPENDÊNCIA DO SÓCIO	292
C.22.1 Instrutores ou gerentes de projeto devem	293
C.22.2 Estudantes ou desenvolvedores devem.....	293
C.23 DIRETRIZ 23 – INTERVALOS PERIÓDICOS.....	293
C.23.1 Instrutores ou gerentes de projeto devem	294
C.23.2 Estudantes ou desenvolvedores devem.....	294

C.24 DIRETRIZ 24 – AVALIAÇÃO DOS TRABALHOS DE OUTRAS DUPLAS.....	294
C.24.1 Instrutores ou gerentes de projeto devem	295
C.24.2 Estudantes ou desenvolvedores devem.....	295

INTRODUÇÃO

Ensino-aprendizagem de programação preocupa os educadores desde os primórdios da computação. Esta preocupação se acentuou na década de 80, quando os computadores se tornaram populares e acessíveis – culminando no aumento da demanda mundial por cursos na área. Entre as décadas de 50 e 80, a preocupação se manteve voltada ao aprendizado de ferramentas específicas (Linguagens de programação utilizadas na época como, por exemplo, Assembler, Cobol, Ada, etc) para trabalhar em um mercado extremamente restrito (JOHNSON e SOLOWAY, 1985). A partir da popularização do PC, inúmeros postos de trabalho surgiram em empresas tradicionais e em empresas iniciantes que vislumbraram o potencial e a importância do software na sociedade da informação. Em consequência disso, e devido à evolução e criação de diversas linguagens de programação comerciais, o foco da educação em computação se voltou ao ensino de conhecimentos mais amplos, para uma quantidade de interessados cada vez maior, no sentido de produzir profissionais aptos a lidar com a complexidade crescente dos softwares e acompanhar as tendências do mercado. Como era de se esperar, as dificuldades no ensino de programação se multiplicaram.

Segundo Proulx (2000, apud Tobar e colaboradores, 2001), essas dificuldades se transformam em uma grande barreira para os estudantes, ocasionando em um grau elevado de reprovações e desistência em disciplinas de programação e, em muitos casos, de evasões em cursos de computação. Os motivos são diversos: a preocupação excessiva com detalhes de sintaxe da linguagem usada (C, Pascal, Java, etc); a falta de uma visão daquilo que se deseja solucionar, de idealizar soluções adequadas, de mapear essas soluções em passos sequenciais e de abstrair o funcionamento dos mecanismos escolhidos; e o estabelecimento de um raciocínio lógico visando à resolução de problemas, com base em um modelo incremental, em relação à complexidade e à estratégia de refinamentos sucessivos.

A literatura mostra que diversas foram as abordagens utilizadas para amenizar este problema. Enquanto alguns autores envidaram esforços em técnicas de aprendizado individual, a exemplo de Tomek (1982), Tomek, Muldner e Khan (1985), Brown (1988), Bently e Kerningham (1991), Stasko e McCrickard (1995), outros como, por exemplo, Whittington (2004), Preston (2005), Matzko e Davis (2006), Salleh, Mendes e Grundy (2010), investiram em métodos colaborativos. Enquanto raros autores se preocuparam em elaborar

métodos específicos para o ensino de programação, diversos outros investiram na construção de ferramentas de apoio automatizadas. São exemplos: PMS (TOMEK, MULDER e KHAN, 1985); PROUST (JOHNSON e SOLOWAY, 1985); VIP (MENDES e MENDES, 1988); LISP-TUTOR (ANDERSON e REISER, 1985); COACH (SELKER, 1994); C-TUTOR (SONG e colaboradores, 1997); PROOGRAMA (GIRAFFA, MARCZAK e ALMEIDA, 2003); APRENDE (CASTRO, CASTRO JR. E MENEZES, 2004); LEARN IN GROUP (FARIA, VILELA e COELLO, 2005); WEB-UNERJOL (FERRANDIN e STEPHANI, 2005); etc.

Fato é, embora encontrados diversos estudos e pesquisas empíricas a respeito e os motivos das dificuldades dos aprendizes em programação de computadores são amplamente conhecidos pelos educadores e facilmente encontrados na literatura, os problemas de ensino-aprendizagem nessa disciplina ainda são percebidos nos cursos de computação e afins.

Procurando motivos para explicar essa constatação, Santos e Costa (2006) esclarecem que ferramentas automatizadas são importantes, mas devem ser cuidadosamente estudadas para sua implantação como apoio ao ensino. Essas ferramentas são eficientes apenas se estiverem atreladas e baseadas em um método específico de ensino. Até então, o que mais se vê é o uso de estratégias isoladas de ensino e ferramentas baseadas em teorias educacionais como, por exemplo, a teoria construtivista de ensino. A exemplo de Borges (2000), poucos são os trabalhos que se ocuparam da produção de métodos específicos de ensino-aprendizagem de programação.

Diante disso, procurou-se explorar os conceitos de métodos de ensino e percebeu-se a importância em trabalhar com objetivos educacionais. A taxionomia de objetivos educacionais proposta por Bloom e colaboradores (1956 e 1964) foi estudada no intuito de desenvolver um método próprio, nomeado como Método Trifásico de Ensino-Aprendizagem (MTEA). Por acreditar no potencial do conflito sócio-cognitivo, propôs-se que o método deve ser implantado através do uso da técnica de Programação em Duplas.

Estudos empíricos foram realizados no curso de Sistemas de Informação e no curso de Engenharia de Produção de uma Universidade da iniciativa privada com o objetivo de verificar a aceitação do MTEA por parte dos aprendizes e de medir sua motivação ao trabalharem sob tal método.

Os objetivos e a motivação do trabalho são os que se seguem.

1.1 OBJETIVO GERAL

Construir um método de ensino-aprendizagem baseado na taxionomia de objetivos educacionais de Bloom e aplicá-lo, experimentalmente, na disciplina “Programação de Computadores” através da técnica de Programação em Duplas.

1.2 OBJETIVOS ESPECÍFICOS

Entre os diversos objetivos específicos que permeiam o trabalho, destacam-se os seguintes:

- Aplicar a taxionomia de objetivos educacionais de Bloom no domínio cognitivo na disciplina “Programação de Computadores” no intuito de gerar conhecimento específico para a construção de ferramentas educacionais voltadas a objetivos específicos;
- Aplicar Programação em Duplas no intuito de mostrar como os objetivos educacionais de Bloom no domínio afetivo podem ser alcançados durante a disciplina de “Programação de Computadores”;
- Levantar informações a respeito da satisfação e motivação dos alunos quando trabalham em duplas nas aulas de laboratório;
- Produzir uma cartilha descrevendo o que podem ser compreendidas como boas atitudes de professores e de aprendizes durante seções de Programação em Duplas com o intuito de tornar eficiente o processo de emparelhamento (Apêndice C) – uma das dificuldades encontradas nesse trabalho está relacionada ao fato de que os alunos não conhecem as regras e boas atitudes para que o trabalho colaborativo seja eficaz.

1.3 PUBLICAÇÕES ORIGINADAS DESTE TRABALHO

- FARIA, E. S. J.; YAMANAKA, K.; TAVARES, J. A.; PINTO, G. H. L.; MELO, L. H. S. Intelligent Software Agents Mediating the Pair Participation in a Distributed Intelligent Pair-Software Development Environment. In: 3rd IEEE International Workshop on Engineering Semantic Agent Systems (ESAS'08) in conjunction with 32nd Annual IEEE International Computer Software and Applications Conference – COMPSAC'08, Turku. 2008.
- FARIA, E. S. J.; YAMANAKA, K.; TAVARES, J. A.; PINTO, G. H. L.; MELO, L. H. S. AIDDES - Distributed Intelligent Pair-Software Development Environment. In: 32nd Annual IEEE International Computer Software and Applications Conference – COMPSAC'08, Turku. 2008.
- FARIA, E. S. J.; YAMANAKA, K.; TAVARES, J. A.; MELO, L. H. S.; PINTO, G. H. L. A Distributed Intelligent Pair-Software Development Tool. In: Workshop em Desenvolvimento Distribuído de Software co-localizado com o Simpósio Brasileiro de Engenharia de Software, Campinas, SP. 2008.
- FARIA, E. S. J.; YAMANAKA, K. How Should The Participants Of The Pair-Programming Process Act? (Guidelines For The Success In Pair Programming). In: VI International Conference on Engineering and Computer Education – ICECE'09, Buenos Aires. 2009.
- COSTA, R. H. P.; FARIA, E. S. J.; YAMANAKA, K. Programação em Duplas no Aprendizado de Programação de Computadores em um Curso de Engenharia de Produção: Um Estudo Empírico. REIC. Revista Eletrônica de Iniciação Científica (Online), v. X, p. 4, 2010.
- FARIA, E. S. J.; YAMANAKA, K.; TAVARES, J. A. XLP – Uma Metodologia Baseada na Programação Extrema Para Ensino de Programação. Aceito no 1º semestre de 2010 para publicação na Revista de Informática Teórica e Aplicada – RITA (Instituto de Informática da Universidade Federal do Rio Grande do Sul).

- FARIA, E. S. J.; YAMANAKA, K. Programação em Duplas: Estado da Arte. Aceito no 2º semestre de 2010 para publicação na Revista Ciências Exatas e Naturais – RECEN (Universidade Estadual do Centro Oeste – UNICENTRO/PR).
- FARIA, E. S. J.; YAMANAKA, K. Extreme Learning of Programming – A Methodology Based on eXtreme Programming for Programming Teaching-Learning. Aceito em 28 de julho de 2010 para publicação no INFOCOMP – Journal of Computer Science (Departamento de Ciência da Computação – Universidade Federal de Lavras).

1.4 MOTIVAÇÃO E JUSTIFICATIVA

As disciplinas de introdução à programação são consideradas fundamentais nos cursos de computação e áreas afins. A Sociedade Brasileira de Computação, por exemplo, promove o Workshop de Ambientes de Apoio à Aprendizagem de Algoritmos e Programação – vinculado ao Simpósio Brasileiro de Informática na Educação (SBIE). A própria existência deste workshop mostra que, mesmo com todas as pesquisas já realizadas na área, as dificuldades na aprendizagem e os altos índices de desistência e repetência continuam existindo nas disciplinas relacionadas (ADEMIR DE JESUS e RAABE, 2009).

Segundo Aharoni (2000), processo de ensino e aprendizado de programação é crucial não apenas porque programar é uma “habilidade prática”, mas também porque processo é o meio pelo qual é formado conhecimento efetivo e conceitual em um esquema bem definido. Tradicionalmente, a comunidade acadêmica atribuiu e ainda tem atribuído muita importância ao tipo de conhecimento conceitual que os estudantes precisam adquirir e a relação hierárquica daquele conhecimento. Isto representa uma ênfase pedagógica no conteúdo, perpetuada inevitavelmente por professores que adotam ou recorrem a tais guias curriculares. Corroborando com as idéias de Bower (2008) e Aharoni (2000), é defendido que o problema do ensino e aprendizado de programação não se encontra no nível de conteúdos, mas sim em um nível de processo que define como os conteúdos são ministrados e que habilidades são exigidas que os estudantes possuam para dominarem o conhecimento nessa área.

Não obstante, durante o levantamento bibliográfico, percebeu-se a carência de modelos de processo colaborativo de software relatados na literatura para fins acadêmicos. Embora tenham sido encontrados diversos estudos empíricos sobre Programação em Duplas, detectou-se que a grande maioria deles não faz referência e não foi baseada em nenhum modelo de processo colaborativo de software.

Concomitante aos motivos anteriores, o alto índice de evasão nas Universidades tem tornado a manutenção de cursos de computação extremamente onerosa. Embora possam ser apontados diversos fatores que colaboram para a evasão dos alunos nesses cursos, a falta de motivação tem papel significativo neste cenário. Será mostrado neste trabalho que o uso de Programação em Duplas aumenta consideravelmente a motivação dos aprendizes durante o curso da disciplina “Programação de Computadores”. Uma vez que a referida disciplina é compreendida como disciplina base e chave em computação, foi percebido que a motivação adquirida durante seu curso pode afetar positivamente a visão que os alunos formam da área com um todo.

Por fim, a indústria de software exige bons e sólidos conhecimentos de técnicas de trabalho colaborativo de seus colaboradores – compreender bem os conceitos do trabalho em equipe pode ser um caminho promissor para aqueles que pretendem construir carreira em informática.

1.5 ESTRUTURA DA TESE

O capítulo 2 apresenta a taxionomia de objetivos educacionais de Bloom no domínio cognitivo e no domínio afetivo. Procurou-se enfatizar: a importância dos objetivos no domínio cognitivo no intuito de avaliar o conteúdo ensinado; e a importância dos objetivos no domínio afetivo como meio de aumentar a motivação dos aprendizes.

O capítulo 3 faz uma explanação da técnica de Programação em Duplas e descreve seus benefícios para a academia e para a indústria de software.

No capítulo 4, é apresentado o estado da arte sobre pesquisas relacionadas ao uso de taxionomias de objetivos educacionais aplicadas ao ensino de programação de computadores. São mostrados outros estudos com objetivos correlatos ou similares ao presente trabalho, enquanto é descrito seu diferencial em relação aos demais.

O capítulo 5 apresenta o Método Trifásico de Ensino-Aprendizagem (MTEA) baseado na taxionomia de objetivos educacionais de Bloom – produzido como fruto desta pesquisa.

O capítulo 6 é constituído de uma aplicação do MTEA. Procurou-se modelar, neste capítulo, os objetivos educacionais nos domínios cognitivo e afetivo para a disciplina “Programação de Computadores”.

O capítulo 7 é destinado ao estudo empírico em que o MTEA foi aplicado na disciplina “Programação de Computadores”, como estudo de caso, em uma Universidade da iniciativa privada.

O capítulo 8 descreve as conclusões e sugestões de trabalhos futuros, enquanto as referências podem ser encontradas em seguida.

Os Apêndices A e B são destinados respectivamente a mostrar alguns exercícios propostos para aplicação durante as fases do MTEA e exercícios de lógica, distribuídos aos alunos no início do semestre letivo, no intuito de fomentar o espírito desafiador da disciplina desde os primeiros contatos com a mesma. O Apêndice C descreve um guia prático que direciona para boas atitudes dos aprendizes e professores em seções de Programação em Duplas.

CAPÍTULO 2

TAXIONOMIA DE OBJETIVOS EDUCACIONAIS

Taxionomia (do grego *taxis* = ordenação e *nomos* = sistema, norma) é todo e qualquer sistema de classificação que possua três características: (a) cumulatividade, (b) hierarquia e (c) eixo comum. Uma taxionomia deve seguir as seguintes regras: cada categoria do sistema de classificação deve abranger as categorias precedentes; no sistema de classificação da taxionomia, uma categoria é superior às que a precedem e inferior às que lhe sucedem; deve existir um alvo comum a todas as categorias que integram a taxionomia (HAMZE, 2009).

A taxionomia de objetivos educacionais de Bloom é considerada como “um sistema destinado a classificar objetivos educacionais” que possui as três características anteriormente descritas. Sua estrutura é composta em uma forma hierárquica que vai do mais simples ao mais elaborado objetivo educacional, proporcionando o desenvolvimento de atividades que vão crescendo em complexidade até atingir os níveis mais altos (PRATA e NASCIMENTO, 2007).

Segundo Bloom e colaboradores (1956, apud DAVIES, 1973), objetivos não são apenas as metas na direção das quais os currículos são estruturados e as instruções são realizadas, mas também as metas que fornecem especificações detalhadas para a elaboração e o uso de técnicas de avaliação. Para os autores, os objetivos fornecem aos professores e aos estudantes “alguma direção envolvendo o conteúdo e o processo mental que é esperado que os aprendizes desenvolvam”.

De acordo com Mager (1962, apud BLOOM e colaboradores, 1964), o objetivo é um intento comunicado por uma especificação que descreve uma mudança proposta no aprendiz - uma especificação sobre a situação do aprendiz quando ele tiver concluído com sucesso a

experiência de aprendizagem. É uma descrição de um padrão de comportamento (desempenho) que se deseja que o aprendiz seja capaz de demonstrar. Em outras palavras, um objetivo é uma especificação de desempenho e responsabilidade; uma meta revestida da completa intenção de ser alcançada.

A principal finalidade da elaboração de uma taxionomia de objetivos educacionais é facilitar a comunicação (BLOOM e colaboradores, 1956). Os autores, portanto, conceberam a taxionomia como um método de favorecer a troca de idéias e materiais entre os especialistas em avaliação, bem como entre outras pessoas vinculadas à pesquisa educacional e ao desenvolvimento de currículos. O seu uso, por exemplo, como auxílio para estabelecer uma definição e classificação precisas de termos vagamente definidos, como “pensamento” e “solução de problemas”, pode capacitar um grupo de instituições acadêmicas a discernir as semelhanças e diferenças entre os objetivos de seus diversos programas educacionais.

Deve-se observar que Bloom não pretendeu classificar metodologias de ensino, modos de relacionamento de professores com alunos ou diferentes tipos de materiais de ensino empregados. Não foi interesse de sua equipe classificar um material específico de ensino ou conteúdos, mas sim classificar o “comportamento esperado” — a maneira na qual os alunos devem agir, pensar ou sentir como resultado de sua participação em qualquer unidade de ensino.

2.1 HISTÓRIA DA CRIAÇÃO DA TAXIONOMIA

Segundo Bloom e colaboradores (1956), a idéia de criação da taxionomia nasceu em uma reunião informal de examinadores universitários, durante a “Convenção da Associação Americana de Psicologia”, em Boston, no ano de 1948. Nesse encontro, se manifestou interesse por um quadro teórico de referência que facilitasse a comunicação entre examinadores. O grupo sentiu que este quadro de referência muito poderia oferecer, oportunizando a troca de idéias e materiais sobre avaliação. Além disso, seria proveitoso no sentido de estimular a pesquisa sobre avaliação e sobre relações entre educação e avaliação. Depois de exaustiva discussão, o grupo concordou que a forma mais adequada para obter esse

quadro de referência seria um sistema de classificação de objetivos, uma vez que estes constituem a base do planejamento do currículo e da avaliação e representam o ponto de partida de muitas pesquisas educacionais.

Esta reunião informal tornou-se a primeira de uma série de reuniões anuais de examinadores universitários. Reunindo-se em Universidades diferentes, cada ano, com algumas modificações em número de seus membros, o grupo examinou problemas envolvidos na organização de um sistema de classificação de objetivos educacionais e também considerou muitos problemas de avaliação e de pesquisa educacional.

Os planos originais da equipe de Bloom objetivavam o desenvolvimento de uma taxionomia completa, constituída de três domínios principais — os domínios cognitivo, afetivo e psicomotor.

O domínio cognitivo inclui aqueles objetivos vinculados à memória ou reconhecimento e ao desenvolvimento de capacidades e habilidades intelectuais. Este domínio é fundamental para a implementação da avaliação no sistema educacional tradicional. Bloom publicou a primeira edição da taxionomia de objetivos educacionais para o domínio cognitivo em 1956.

O domínio afetivo inclui objetivos que descrevem mudanças de interesse, atitudes e valores e o desenvolvimento de apreciações e ajustamento adequado. Segundo Bloom e colaboradores (1956), objetivos neste domínio não apresentam uma formulação precisa e “não é fácil formular objetivos que descrevam comportamentos nesta área, uma vez que os sentimentos e as emoções interiores ou não manifestos são tão significativos quanto os comportamentos manifestos ou abertos”. Apenas em 1964 a equipe de Benjamim Bloom conseguiu publicar a primeira edição da taxionomia de objetivos educacionais para o domínio afetivo.

O terceiro domínio diz respeito à área das habilidades manipulativas ou motoras. Embora se reconheça a existência deste domínio, encontravam-se tão poucas realizações a este respeito na escola secundária e nas faculdades que Bloom e sua equipe nunca desenvolveram uma taxionomia para a área psicomotora, porém, outros especialistas o fizeram – é o caso de Simpson (1966) e Harrow (1972).

Os anos se passaram e a taxionomia de Bloom foi utilizada como apoio pedagógico nas mais diversas ciências. Como resposta às críticas de alguns pesquisadores, em 2001, Anderson e colaboradores revisaram a taxionomia de Bloom para o domínio cognitivo em uma publicação denominada “Uma Taxionomia para Aprendizado, Ensino e Avaliação: Uma Revisão da Taxionomia de Objetivos Educacionais de Bloom”.

2.2 O DOMÍNIO COGNITIVO

Esta área pode também ser explicitada pelo fato de abranger comportamentos que incluem memória, raciocínio, solução de problemas, formação de conceitos e, numa extensão limitada, pensamento criador (BLOOM e colaboradores, 1956).

Pensando nas características inerentes de uma taxionomia, Bloom construiu sua taxionomia de objetivos educacionais para o domínio cognitivo de forma que a ordem dos termos correspondesse a uma certa ordem real entre os fenômenos representados por estes termos. A estrutura da taxionomia proposta para o domínio cognitivo é constituída de seis classes ou categorias principais: 1) Conhecimento; (2) Compreensão; (3) Aplicação; (4) Análise; (5) Síntese; e (6) Avaliação. Essas classes foram definidas de modo que os objetivos categorizáveis em uma classe compreendem e se baseiam em comportamentos incluídos nas classes precedentes do esquema.

Como a taxionomia é hierárquica, cada nível é incluído pelos níveis mais altos. Em outras palavras, um estudante que trabalha no nível “aplicação” também já dominou o material no nível de “conhecimento” e “compreensão”. Durante muitos anos, essa estrutura foi comparada a uma escadaria. Assim, o professor deve propor ao aluno “escalar” desde o primeiro “degrau” até os últimos níveis. A figura 2.1 ilustra como os aprendizes avançam nas categorias do conhecimento segundo o domínio cognitivo.

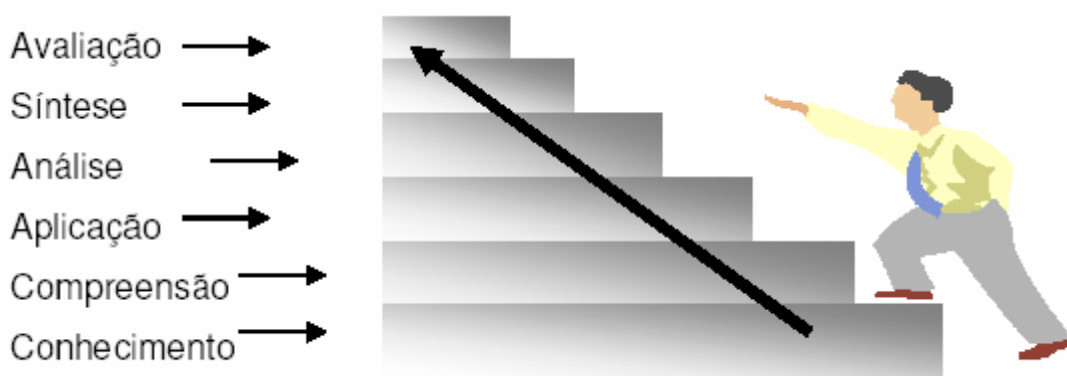


Figura 2.1 – Avançando nas Categorias do Conhecimento Segundo o Domínio Cognitivo de Bloom.

2.2.1 Conhecimento

O conhecimento, de acordo com Bloom e colaboradores (1956), se relaciona à evocação de específicos e universais, de métodos e processos, ou de um padrão, estrutura ou composição. Evocar, neste sentido, envolve mais do que recordar o material apropriado. Os objetivos neste nível de conhecimento enfatizam, em maior escala, os processos psicológicos da memória.

O conhecimento de específicos refere-se a tipos de informação ou conhecimentos que são isoláveis ou que podem ser evocados separadamente, enquanto o conhecimento de universais e abstrações acentua as inter-relações e configurações nas quais a informação pode ser organizada e estruturada. São exemplos de conhecimentos específicos em programação: conhecimento da sintaxe de uma linguagem de programação; conhecimento das estruturas de controle (sequenciais, condicionais e de repetição); conhecimentos das funções matemáticas disponíveis em uma linguagem; entre outros. Exemplos de conhecimentos universais, por sua vez, envolvem: conhecimentos sobre os sistemas de numeração; teoria dos conjuntos; gramática; autômatos; entre outros.

Segundo Bloom e colaboradores (1956), embora se reconheça a presença do conhecimento nas categorias fundamentais e mais complexas da taxionomia (Compreensão, Aplicação, Análise, Síntese e Avaliação), a categoria “conhecimento” difere das demais pelo

fato de que a evocação é o principal processo psicológico nela envolvido, já que nessas outras categorias a evocação é somente um aspecto de processos muito complexos, como os de relacionamento, julgamento e reorganização.

2.2.1.1 Conhecimento de Específicos

Refere-se à evocação de unidades de informação consideradas específicas e que podem ser isoladas. Este material pode ser entendido como os elementos a partir dos quais formas de conhecimento mais complexas e abstraídas são elaboradas.

2.2.1.1.1. Conhecimento de Terminologia

Conhecimento dos referentes para símbolos específicos (verbais e não verbais). Isto pode incluir conhecimento dos referentes simbólicos mais geralmente aceitos, da variedade de símbolos que pode ser usada para um único referente, ou do referente mais apropriado a um determinado uso de um símbolo. São exemplos: a definição de termos técnicos em computação, descrevendo seus atributos, propriedades ou relações entre eles; a familiaridade com as palavras chave de uma linguagem de programação; conhecimento e vocabulário suficientes para dialogar com um desenvolvedor sobre possíveis problemas sintáticos em um código fonte; conhecimento dos termos e conceitos peculiares de programação orientada a objetos; entre outros.

2.2.1.1.2. *Conhecimento de Fatos Específicos*

Conhecimento de datas, acontecimentos, pessoas, lugares, etc. Isto pode incluir informações muito precisas e específicas, tais como a data específica ou a magnitude exata de um fenômeno. Pode abranger também informações aproximadas ou relativas, tais como um período de tempo aproximado ou a ordem geral de magnitude de um fenômeno. São exemplos: a evocação de fatos principais a respeito da evolução das linguagens ou dos paradigmas de programação; a história evolutiva dos computadores; informações específicas sobre as características das diversas versões da linguagem Java; entre outros.

2.2.1.2 Conhecimento de Maneiras e Meios de Tratar com Específicos

Conhecimento das maneiras de organizar, estudar, julgar e criticar. Isto inclui os métodos de indagação, as sequências cronológicas e os padrões de julgamento dentro de um campo, tanto quanto os padrões de organização através dos quais as áreas dos próprios campos são determinadas e organizadas internamente. Este conhecimento se encontra em um nível intermediário de abstração entre conhecimento específico, de um lado, e conhecimento de universais, de outro. Não exige do estudante tanta atividade no uso dos materiais quanto uma percepção passiva da natureza dos mesmos.

2.2.1.2.1. *Conhecimento de Convenções*

Conhecimento das maneiras características de tratar e apresentar idéias e fenômenos. Para efeito de comunicação e consistência, os que trabalham em determinado campo empregam usos, estilos, práticas e formas que melhor se ajustam a seus propósitos e/ou que parecem se adaptar melhor aos fenômenos de que tratam. Ainda que essas formas e convenções tenham probabilidade de se estabelecerem em bases arbitrárias, acidentais ou

autoritárias, são mantidas por causa da concordância ou cooperação geral dos indivíduos interessados no assunto, nos fenômenos ou no problema. São exemplos: familiaridade com as formas e convenções dos principais estilos e padrões de programação; conscientização dos alunos sobre a forma e uso correto das estruturas de repetição; consciência das regras usuais de escrita de comentários, identificadores e constantes; conhecimento das regras de pontuação de uma linguagem (onde usar “:”, “;”, etc); entre outros.

2.2.1.2.2. Conhecimento de Tendências e Sequências

Conhecimento dos processos, direções e movimentos dos fenômenos em relação ao tempo. Inclui conhecimento de tendências que indicam os relacionamentos entre uma série de acontecimentos específicos localizados em momentos distanciados. Abrange também representações de processos que envolvem tempo e inter-relações causais entre uma série de eventos específicos. São exemplos: compreensão da importância do desenvolvimento dos paradigmas de programação; conhecimento das tendências na utilização de linguagens de programação não proprietárias; conhecimento das tendências do desenvolvimento de software voltado para a Internet; conhecimento das influências das primeiras linguagens de programação nas linguagens atuais; entre outros.

2.2.1.2.3. Conhecimento de Classificações e Categorias

Conhecimento das classes, conjuntos, divisões e disposições que são considerados fundamentais para um certo campo de assunto, propósito, argumento ou problema. Na medida em que uma área de conhecimento, problema ou tópico torna-se bem desenvolvida, as pessoas que nela trabalham julgam como útil a elaboração de classificações que lhes auxiliem a estruturar e sistematizar os fenômenos. São exemplos: conhecimento dos paradigmas de programação e suas características; conhecimento das estruturas de controle (sequenciais, condicionais e de repetição); conhecimento a respeito dos tipos primitivos de dados; entre outros.

2.2.1.2.4. *Conhecimento de Critérios*

Conhecimento dos critérios pelos quais fatos, princípios, opiniões e conduta são testados e julgados. São exemplos: familiaridade com critérios de julgamento de qualidade de programas (por exemplo, critérios de legibilidade, de eficiência, de corretude, etc). É importante destacar que, em computação, esses critérios são amplamente conhecidos como métricas de software; familiaridade com critérios de avaliação de documentos de projeto de software (por exemplo, obediência a padrões pré-definidos, legibilidade, validação e verificação de funcionalidades, etc); entre outros.

2.2.1.2.5. *Conhecimento de Metodologia*

Conhecimento dos métodos de indagação, técnicas e procedimentos empregados num campo específico de assunto, tanto quanto aqueles empregados na investigação de problemas e fenômenos determinados. A ênfase nesta categoria está mais relacionada ao conhecimento do método pelo indivíduo do que em sua capacidade para usá-lo. O conhecimento de metodologia é importante porque, antes que o aluno se envolva em uma investigação, pode ser interessante e de grande valia conhecer os métodos e técnicas utilizados em outros trabalhos similares. Num estágio posterior à sua investigação, o aluno pode estabelecer as relações entre os métodos que empregou e os métodos usados por outras pessoas. São exemplos: conhecimento de metodologias de desenvolvimento de software; conhecimento de processos de software; conhecimento de padrões de projeto de software; entre outros.

2.2.1.3 *Conhecimento dos Universais e Abstrações em um Certo Campo*

Conhecimento dos principais esquemas e padrões pelos quais as idéias e os fenômenos são organizados. Estas são as grandes estruturas, teorias e generalizações que dominam um

campo de assunto ou que, em geral, são bastante usadas no estudo de fenômenos ou na solução de problemas. Este conhecimento se encontra nos níveis mais elevados de abstração e complexidade (BLOOM e colaboradores, 1956).

2.2.1.3.1. Conhecimento de Princípios e Generalizações

Conhecimento de abstrações determinadas que sintetizam as observações de fenômenos. São as abstrações de valores para explicar, descrever, predizer ou determinar a ação ou direção mais apropriada e pertinente a ser adotada em uma determinada situação. São exemplos: conhecimento de funções matemáticas para utilização de linguagens do paradigma de programação funcional; conhecimento de lógica proposicional para utilização de linguagens do paradigma lógico; entre outros.

2.2.1.3.2. Conhecimento de Teorias e Estruturas

Conhecimento de princípios e generalizações junto com suas inter-relações, que apresentam uma visão clara, harmoniosa e sistemática de um fenômeno, problema ou campo complexo. São as formulações mais abstratas, e podem ser usadas para demonstrar a inter-relação e organização de uma extensa série de específicos. Esta categoria difere da anterior, pois a ênfase aqui recai sobre um corpo de princípios e generalizações inter-relacionados para formar uma teoria ou estrutura, enquanto os princípios e generalizações de que trata a categoria anterior são particulares e não estão necessariamente relacionados entre si. São exemplos: conhecimento dos princípios do paradigma de programação funcional; entendimento das inter-relações dos princípios dos paradigmas de programação; entre outros.

2.2.2 Capacidades e Habilidades Intelectuais

As capacidades e habilidades dizem respeito a modos organizados de operação e a técnicas generalizadas para tratar com materiais e problemas. Os materiais e problemas podem ser de tal natureza que pouca ou nenhuma informação especializada e técnica seja requerida. Tal informação, na medida em que é requerida, pode ser pressuposta como parte do acervo geral de conhecimento do indivíduo.

Outros problemas podem exigir informação técnica e especializada em um nível bastante alto, de modo que sejam requeridos o conhecimento específico e a habilidade em tratar com o problema e os materiais. Os objetivos envolvendo as capacidades e habilidades dão ênfase aos processos mentais de organização e reorganização de material para atingir um certo propósito. Os materiais podem ser dados ou evocados.

2.2.2.1 Compreensão

Segundo Bloom e colaboradores (1956), este representa o nível mais baixo de entendimento. Refere-se a um tipo de entendimento ou apreensão tal que o indivíduo conhece o que está sendo comunicado e pode fazer uso do material ou idéia que está sendo comunicada, sem necessariamente relacioná-la a outro material ou perceber suas implicações mais completas. Pode ser classificado em Tradução, Interpretação ou Extrapolação.

2.2.2.1.1. Tradução

Pode ser definida basicamente como compreensão, conforme é evidenciada pelo cuidado e precisão com que a comunicação é transformada ou vertida de uma língua ou forma de comunicação a outra. A tradução é julgada com base na fidelidade e precisão; isto é, na

medida em que o material na comunicação original é preservado, ainda que a forma da comunicação tenha sido alterada. São exemplos: a capacidade para compreender enunciações não literais (metáfora, simbolismo); habilidade para transformar material verbal matemático em expressões simbólicas e vice-versa (sistema de equações); capacidade de compreender ou criar uma abstração, tal como um princípio geral, por meio de ilustrações e exemplos (fila, pilha, lista, árvore, etc); entre outros.

2.2.2.1.2. Interpretação

Refere-se à explicação ou síntese de uma comunicação. Enquanto a tradução envolve uma representação objetiva de uma comunicação, parte por parte, a interpretação subentende uma reordenação, redistribuição ou nova visão material. A qualidade da interpretação está diretamente relacionada aos conhecimentos prévios do indivíduo sobre o tema chave da comunicação. São exemplos: capacidade de ler e interpretar algoritmos; capacidade de construir um programa eficiente em uma linguagem definida a partir da interpretação de um algoritmo; entre outros.

2.2.2.1.3. Extrapolação

Refere-se à extensão das direções ou tendências além dos dados fornecidos, para determinar implicações, consequências, corolários, efeitos, etc., que estão de acordo com as condições descritas na comunicação original.

A extrapolação requer que o leitor perceba com clareza os limites dentro dos quais uma comunicação se encontra e pode ser entendida. Em todos os casos, praticamente, o leitor deve reconhecer que uma extrapolação pode somente ser uma inferência com algum grau de probabilidade – sendo rara a certeza no que se refere à extrapolação. São exemplos: habilidade para prever a continuação das tendências das linguagens de programação; capacidade para tirar conclusões a partir de efeitos colaterais em software e expressá-las com

efetividade; capacidade para calcular ou prever consequências de linhas ou blocos de linhas de código; entre outros.

2.2.2.2 Aplicação

Está relacionada ao uso de abstrações em situações particulares e concretas. As abstrações podem apresentar-se sob a forma de idéias gerais, regras de procedimentos ou métodos generalizados. As abstrações podem também ser princípios técnicos, idéias e teorias, que devem ser recordados e aplicados. Nessa categoria, o aluno deve ser capaz (em um problema ainda não visto por ele) de aplicar as abstrações apropriadas sem que lhe tenha sido sugerido quais são estas abstrações ou sem que lhe seja ensinado como usá-las naquela situação. São exemplos: aplicação dos conceitos de estruturas de dados (fila, pilha e lista) ao construir um sistema operacional; capacidade de aplicar a teoria de grafos em situações práticas; aplicação da teoria de logaritmo na produção de algoritmos eficientes (como, por exemplo, método de ordenação *Quicksort*, método de pesquisa binária, etc); entre outros.

2.2.2.3 Análise

Diz respeito ao desdobramento de uma comunicação em seus elementos ou partes constituintes, de modo que a hierarquia relativa de idéias é tornada clara e/ou as relações entre as idéias expressas são tornadas explícitas. Tais análises têm a intenção de esclarecer a comunicação, de indicar como a comunicação é organizada e a maneira pela qual consegue transmitir seus efeitos, assim como sua base e disposição.

2.2.2.3.1. Análise de Elementos

Identificação dos elementos incluídos numa comunicação. São exemplos: a capacidade para reconhecer suposições não enunciadas; habilidade para distinguir fatos de hipóteses; habilidade para identificar motivos de falhas em algoritmos; capacidade de analisar e identificar as classes, atributos e métodos necessários à produção de um software sob o paradigma orientado a objetos; entre outros.

2.2.2.3.2. Análise de Relações

Uma vez identificados os diversos elementos de uma comunicação, o leitor ainda tem diante de si a tarefa de determinar as principais relações entre os elementos, como também as relações entre as diversas partes que a constituem. Às conexões e interações entre elementos e partes de uma comunicação dá-se o nome de Análise de Relações. São exemplos: capacidade para verificar a coerência de hipóteses com certas informações e suposições; habilidade para compreender as inter-relações entre blocos de código fonte; capacidade de compreender e estabelecer as inter-relações necessárias entre os objetos de um código fonte orientado a objetos; entre outros.

2.2.2.3.3. Análise dos Princípios Organizacionais

Refere-se à organização, à disposição sistemática e à estrutura que conservam a comunicação unificada. Isto inclui a estrutura “explícita”, bem como a “implícita”. Abrange as bases, a disposição necessária e a mecânica, que fazem da comunicação uma unidade. Segundo Bloom e colaboradores (1956), a tarefa de analisar a estrutura e a organização de uma comunicação se encontra em nível bastante difícil e complexo. O autor de uma comunicação raramente indica de forma explícita os princípios de organização empregados e, em algumas situações, não tem uma clara concepção de que os utilizou. São exemplos:

capacidade para reconhecer forma e padrão de codificação em um programa fonte; capacidade para reconhecer as técnicas gerais usadas em programas cuja qualidade é considerada alta; entre outros.

2.2.2.4 Síntese

Está relacionada à combinação de elementos e partes, de modo a formar um todo. Isto envolve o processo de trabalhar com peças, partes, elementos, etc., dispondo-os e combinando-os para que constituam um padrão ou estrutura que antes não estava evidente. Em geral, implica em uma recombinação de partes examinadas em experiência anterior com materiais novos, os quais são reorganizados em um todo novo e bem integrado. Segundo Bloom e colaboradores (1956), esta é a categoria do domínio cognitivo em que se proporcionam ao aluno maiores oportunidades de desenvolver um comportamento criador. No entanto, deve-se assinalar, as expressões criadoras neste nível não são completamente livres, pois, em geral, o aluno necessita trabalhar dentro dos limites impostos por determinados problemas, materiais ou estruturas teóricas e metodológicas.

2.2.2.4.1. Produção de uma Comunicação Singular

O desenvolvimento de uma comunicação em que o escritor ou quem fala tenta transmitir idéias, sentimentos e/ou experiências aos outros. Esta subcategoria abrange objetivos cuja ênfase principal reside na comunicação. Os principais fatores que controlam ou limitam tarefas deste tipo são os seguintes: os efeitos a serem alcançados; as características pessoais dos que são influenciados por esses efeitos; o meio específico pelo qual o estudante se expressa; e as idéias e experiências particulares que o aluno desenvolve ou deseja comunicar. São exemplos: habilidade para codificar, usando uma excelente organização e combinação de instruções; capacidade de construir código eficaz e eficiente; Capacidade para escrever um algoritmo inovador para resolver um problema antigo; entre outros.

2.2.2.4.2. Produção de um Plano ou Indicação de um Conjunto de Operações

Refere-se ao desenvolvimento de um plano de trabalho ou a proposta de um plano de operações. O plano deverá satisfazer requisitos da tarefa, que podem ser dados ao estudante ou que ele mesmo pode desenvolver. São exemplos: capacidade de projetar uma solução de software; capacidade de modelar um software usando linguagem de modelagem apropriada como, por exemplo, DFD, DER, diagramas da UML, etc; capacidade para projetar uma interface de um software com base em certas especificações; entre outros.

2.2.2.4.3. Derivação de um Conjunto de Relações Abstratas

O desenvolvimento de um conjunto de relações abstratas – seja para classificar ou explicar dados ou fenômenos específicos, seja para a dedução de proposições e relações a partir de um conjunto de proposições básicas ou representações simbólicas. São exemplos: capacidade para formular hipóteses apropriadas em uma solução de software, baseadas na análise de fatores envolvidos, e para modificar tais hipóteses à luz de novos fatores e considerações; capacidade para fazer descobertas e generalizações matemáticas; capacidade para formular uma abstração de um problema complexo baseada em outras abstrações e aplicá-la em um algoritmo inovador; entre outros.

2.2.2.5 Avaliação

Avaliação pode ser definida como o processo de julgamento a respeito do valor do “material” e dos “métodos” para certos propósitos. Esses julgamentos podem ser quantitativos e qualitativos acerca da medida em que “material” e “métodos” satisfazem os critérios. Implica no uso de critérios e padrões que permitem apreciar o grau de precisão, efetividade, economia ou suficiência de pormenores. Os critérios podem ser aqueles determinados pelo

estudante ou aqueles que lhe são dados. Os julgamentos podem ser feitos em termos de evidência interna e/ou em termos de critérios externos.

2.2.2.5.1. Julgamentos em Termos de Evidência Interna

Refere-se à avaliação da precisão de uma comunicação a partir de evidências tais como precisão lógica, consistência e outros critérios internos. São exemplos: a capacidade para indicar incongruências lógicas em algoritmos ou em blocos de instruções; capacidade para aplicar certos critérios (baseados em padrões internos) no julgamento de um algoritmo; entre outros.

2.2.2.5.2. Julgamentos em Termos de Critérios Externos

Avaliação de material com referência a critérios selecionados ou evocados. Os critérios podem representar fins a serem atingidos; técnicas, regras ou normas segundo os quais os trabalhos são julgados; ou a comparação do trabalho com outros trabalhos. São exemplos: a comparação entre algoritmos a partir de critérios definidos; julgando por padrões externos, a capacidade para comparar um algoritmo de ordenação inovador com os algoritmos de ordenação mais eficientes (por exemplo, *Quicksort*, *Mergesort*, etc); habilidade de reconhecer e apreciar valores envolvidos em linhas de código alternativas; capacidade para avaliar criticamente métricas de qualidade de software; entre outros.

2.3 O DOMÍNIO AFETIVO

O domínio afetivo inclui objetivos que descrevem mudanças de interesse, atitudes e valores e o desenvolvimento de apreciações e ajustamento adequado. Pode também ser

explicado como “os objetivos que enfatizam uma tonalidade de sentimento, uma emoção ou um grau de aceitação ou rejeição”. Os objetivos afetivos variam desde a intenção simples até fenômenos selecionados, até qualidades de caráter e de consciência complexas, mais internamente consistentes (BLOOM e colaboradores, 1964).

Em sua busca por descrever os objetivos educacionais para o domínio afetivo, Bloom evidenciou que o trabalho de avaliação, em relação a objetivos afetivos, era marginal e somente feito quando uma questão muito insistente era levantada pelo corpo docente ou quando alguém desejasse fazer pesquisa “educacional”. É importante prestar atenção neste aspecto no intuito de perceber que mesmo após anos de evolução na educação, muitos professores temem avaliar seus aprendizes sob conceitos afetivos.

Por outro lado, não é inteiramente correto subentender que a avaliação da consecução de objetivos afetivos esteja completamente ausente das atividades regulares de escolas e professores. Muitos professores estão alertas para indícios de interesses e atitudes desejáveis e desenvolvimento do caráter. Todavia, a maior parte disto é a observação de características inusitadas ou desenvolvimentos dramáticos quando se impõem à atenção do professor.

O fato que pode explicar a histórica dificuldade em avaliar os aprendizes sob o domínio afetivo pode estar diretamente relacionado ao domínio cognitivo. Segundo Bloom e colaboradores (1964), o aproveitamento cognitivo é considerado como um “jogo limpo” para propósitos de classificação. Os exames podem incluir uma grande extensão de tipos de objetivos cognitivos, e os professores e examinadores hesitam pouco ao atribuir a um estudante uma nota A ou F, com base em sua realização nesses exames de aproveitamento cognitivo. Ao contrário, muitos professores e examinadores não consideram apropriado classificar os estudantes relativamente a seus interesses, atitudes ou desenvolvimento de caráter.

Uma parte considerável da hesitação no uso de medidas afetivas para propósitos de classificação provém da inadequação das técnicas de apreciação e da facilidade com que um estudante pode explorar a sua capacidade em detectar as respostas que serão premiadas e as respostas que estarão sujeitas à penalidade. Em contraste, pressupõe-se que um estudante que responde de uma maneira desejável, numa medida cognitiva, na verdade, possui a competência da qual essa é uma amostra.

Fato é que avaliação sob o domínio afetivo é muito questionado por uns e muito defendido por outros. Embora não seja interesse deste trabalho defender ou questionar a avaliação sob esse aspecto, o domínio afetivo se faz importante no sentido de promover a aceitação dos alunos e consequentemente o aprendizado de conteúdos em programação de computadores.

Prosseguindo, o processo de aprendizado sob o domínio afetivo foi denominado por Bloom como Internalização. O processo de Internalização inicia quando a atenção do estudante é captada por algum fenômeno, característica ou valor. À medida que presta atenção ao fenômeno, característica ou valor, ele o diferencia de outros, presentes no campo perceptual. Com a diferenciação, vem uma procura do fenômeno, à medida que lhe agrega significação emocional e vem a valorizá-lo. Conforme o processo se desdobra, ele relaciona este fenômeno a outros fenômenos, que também têm valor. Esta resposta é suficientemente frequente, de forma que vem a reagir regularmente, quase automaticamente, a ele e a outras coisas como ele. Finalmente, os valores são inter-relacionados numa estrutura ou visão do mundo, que ele leva como uma “direção” para novos problemas.

Mesmo a partir desta descrição abstrata, pode ser percebido que o processo de internalização representa uma modificação contínua do comportamento, desde o indivíduo estar consciente de um fenômeno, até uma perspectiva difundida da vida, que influencia todas as suas ações (BLOOM e colaboradores, 1964).

A estrutura da taxionomia proposta para o domínio afetivo é constituída de cinco classes principais: 1) Acolhimento (atenção); (2) Resposta; (3) Valorização; (4) Organização; e (5) Caracterização por um valor ou complexo de valores.

2.3.1 Acolhimento (atenção)

Neste nível de objetivo, é esperado que o aprendiz seja sensibilizado pela existência de certos fenômenos e estímulos, isto é, que esteja disposto a acolhê-los ou prestar atenção a eles. Este é evidentemente o primeiro passo, se o aprendiz está apropriadamente orientado a

aprender o que o professor pretende que ele faça. Indicar que este é o degrau inferior da escada, entretanto, absolutamente não implica que o professor está iniciando de uma estaca zero. Por causa da experiência prévia (formal ou informal), o estudante traz a cada situação um ponto de vista ou predisposição, que pode facilitar ou pôr obstáculos a seu reconhecimento dos fenômenos, aos quais o professor está tentando fazê-lo sensível (BLOOM e colaboradores, 1964).

A categoria de Acolhimento está dividida em três subcategorias, para indicar três níveis diferentes de prestar atenção ao fenômeno. Embora os pontos de divisão entre as subcategorias sejam arbitrários, as subcategorias representam um *continuum* (relacionamento de continuidade). A partir de uma posição ou papel extremamente passivo, de parte do aprendiz, em que a única responsabilidade para a evocação do comportamento permanece com o professor, para “prender” a atenção do estudante, o continuum se estende até um ponto em que o aprendiz dirige a sua atenção, pelo menos num nível semiconsciente, para os estímulos preferidos.

2.3.1.1 Percepção

Segundo Bloom e colaboradores (1964), “Percepção” está relacionada à mera consciência de alguma coisa – que tome em consideração uma situação, fenômeno, objeto ou conjunto de acontecimentos. A percepção não subentende uma avaliação das qualidades ou da natureza do estímulo. Pode haver um simples dar-se conta, sem discriminação ou reconhecimento das características objetivas do objeto, ainda que essas características devam ser supostas como tendo um efeito. O indivíduo pode não ser capaz de verbalizar os aspectos do estímulo que causam a percepção. São exemplos: o aprendiz desenvolve a percepção de fatores que identificam o estilo de codificação em um algoritmo; o aprendiz desenvolve alguma consciência da estrutura básica de um algoritmo; o aprendiz desenvolve a consciência de que lhe será exigido raciocínio e abstração para desenvolver algoritmos; entre outros.

2.3.1.2 Disposição para Receber

Em um nível mínimo, essa subcategoria descreve o comportamento de estar disposto a tolerar um estímulo dado, não o evitando. Assim como na “Percepção”, envolve uma neutralidade ou julgamento pendente em relação ao estímulo. Neste nível do *continuum*, o professor está interessado em que o estudante, na pior das hipóteses, não esteja ativamente evitando prestar atenção em uma explanação. Na melhor das hipóteses, o aprendiz está disposto a notar o fenômeno e dar-lhe a sua atenção. São exemplos: o aprendiz presta (cuidadosa) atenção quando o seu parceiro ou o professor fala; o aprendiz se conscientiza da importância de participar como piloto e navegador durante uma seção de emparelhamento; entre outros.

2.3.1.3 Atenção Controlada ou Seletiva

Em um nível mais elevado, o interesse é um novo fenômeno, a diferenciação de um certo estímulo em figura e fundo, em um nível consciente ou talvez semiconsciente. A percepção ainda é sem avaliação e o estudante pode não saber os termos técnicos ou símbolos, com os quais descrevê-lo corretamente ou precisamente, para os outros. Existe um elemento do controle da atenção do aprendiz, segundo essa subcategoria, de maneira que o estímulo preferido é selecionado e lhe é dada atenção, apesar de estímulos concorrentes e que podem distrair. São exemplos: o aprendiz lê os capítulos de um livro de programação (escolhidos por ele mesmo); o aprendiz percebe características de qualidade em um algoritmo pronto; o aprendiz realiza buscas na Internet sobre algoritmos que resolvem um determinado problema complexo; entre outros.

2.3.2 Resposta

Neste nível, o que interessa são respostas que ultrapassam o meramente atentar para o fenômeno. O estudante está suficientemente motivado e ativamente prestando atenção. Como uma primeira etapa, num processo de “aprender fazendo”, o estudante se compromete em alguma medida com os fenômenos subentendidos. Segundo Bloom e colaboradores (1964), esta é a categoria que muitos professores acreditam que melhor descreve seus objetivos de “interesse”. Mais comumente, usa-se o termo para indicar o desejo de que um aprendiz se torne suficientemente envolvido ou compromissado com um assunto, fenômeno ou atividade, a ponto de procurar e obter satisfação, trabalhando ou se entregando ao mesmo.

2.3.2.1 Aquiescência na Resposta

Pode-se usar a palavra “obediência” ou “submissão”, para descrever este comportamento. Como ambos estes termos indicam, existe uma passividade por parte do aprendiz, na medida em que se inicia o comportamento, e o estímulo que é requerido para este comportamento não é sutil. Submissão, talvez, seja um termo melhor do que obediência, desde que existe mais do elemento de reação a uma sugestão e menos de implicação de resistência ou de consentimento relutante. O estudante provê a resposta, mas não aceitou completamente a necessidade de fazê-lo. São exemplos: disposição para ler os enunciados de uma lista de exercícios; disposição para realizar buscas na Internet sobre conceitos de lógica no intuito de compreender sua importância para o sucesso do aprendizado durante o curso de programação; entre outros.

2.3.2.2 Disposição para Responder

A chave, para este nível, está no termo “disposição”, com a sua capacidade para atividade voluntária. Existe a implicação de que o aprendiz esteja suficientemente compromissado para manifestar o comportamento, que ele assim o faz não precisamente por causa de um medo de punição, mas “por si mesmo”, ou voluntariamente. Pode ser útil notar que o elemento de resistência ou de consentimento relutante, que está possivelmente presente, no nível prévio, é aqui substituído por consentimento, ou procedência da própria escolha do indivíduo. São exemplos: o aprendiz resolve as listas de exercícios propostas; o aprendiz discute com seu parceiro sobre construções algorítmicas (combinações de instruções) para a solução de um problema; entre outros.

2.3.2.3 Satisfação na Resposta

O elemento adicional nesse nível é que o comportamento é acompanhado por um sentimento de satisfação, uma resposta emocional, geralmente de prazer, gosto ou satisfação. A inclusão desta subcategoria, neste ponto, serve ao propósito pragmático de lembrar da presença do componente emocional e de seu valor, na construção de comportamentos afetivos, embora o componente emocional pode ocorrer gradativamente em cada uma das categorias do domínio afetivo. São exemplos: satisfação em construir soluções algorítmicas; o aprendiz encontra prazer em procurar novas abordagens para construir um algoritmo; o aprendiz obtém prazer em desenvolver programas em dupla; entre outros.

2.3.3 Valorização

O comportamento categorizado neste nível é suficientemente consistente e estável para assumir as características de uma crença ou de uma atitude. O aprendiz manifesta este comportamento com bastante consistência, em situações apropriadas, que vem a ser percebido como adotando um valor. Neste nível, o interesse se encontra na internalização de um conjunto de valores especificados, ideais. Vistos sob outro ponto de vista, os objetivos classificados aqui constituem a matéria-prima da qual a consciência do indivíduo é desenvolvida no controle ativo do comportamento.

Um elemento importante do comportamento, caracterizado por valorização, é que é motivado, não pelo desejo de se submeter ou obedecer, mas pelo cometimento do indivíduo com o valor subjacente, que orienta o comportamento.

Este conceito abstrato de valor é em parte um resultado da própria valorização e avaliação do indivíduo, porém é muito mais um produto social, que foi vagarosamente internalizado ou aceito e veio a ser usado pelo estudante, como seu próprio critério de valor.

2.3.3.1 Aceitação de um Valor

Neste nível, o interesse é a atribuição de um valor a um fenômeno, comportamento, objeto, etc. O termo “crença”, que é definido como a aceitação emocional de uma proposição ou doutrina com base no que a pessoa implicitamente considera um fundamento adequado, descreve muito bem o que pode ser julgado a característica dominante aqui. Aprendizes têm graus variáveis de certeza. Neste nível mais baixo de Valorização, o interesse está voltado para os níveis mais baixos de certeza. É uma posição parecida com uma “tentativa” (BLOOM e colaboradores, 1964).

Uma das características mais distintas deste comportamento é a consistência de resposta à classe de objetos, fenômenos, etc., com os quais a crença ou atitude está

identificada. É bastante consistente, de maneira que a pessoa seja percebida pelos outros como adotando a crença ou valor. São exemplos: desejo contínuo de desenvolver a capacidade de escrever algoritmos eficientes; o aprendiz estuda os critérios que qualificam bons programas e procura aplicá-los em todas as construções algorítmicas; entre outros.

2.3.3.2 Preferência por um Valor

O suprimimento para esta subdivisão surgiu de um sentimento de que havia objetivos que expressavam um nível de internalização, entre a mera aceitação de um valor e cometimento ou convicção, na conotação usual de profundo envolvimento, em uma área. O comportamento, neste nível, subentende não apenas a aceitação de um valor, até o ponto de estar disposto a ser com ele identificado, mas o indivíduo está suficientemente comprometido com o valor, para buscá-lo, procurá-lo, querê-lo. São exemplos: o aprendiz assume responsabilidade em incentivar os colegas de sala a desenvolverem seus trabalhos extraclasse em duplas; o aprendiz examina deliberadamente uma variedade de pontos de vista sobre algoritmos, tendo em vista a formação de opiniões a respeito dos mesmos; o aprendiz se responsabiliza em incentivar os colegas a resolver as listas de exercícios; entre outros.

2.3.3.3 Cometimento

Este nível envolve um grau mais alto de certeza. As idéias de “convicção” e de “certeza sem sombra de dúvida” ajudam a transmitir mais o nível de comportamento pretendido. Em alguns casos, pode estar relacionado com a fé, no sentido de ser uma aceitação emocional firme de uma crença sobre fundamentos admitidamente não-rationais. Lealdade a uma posição, um grupo ou uma causa também pode ser classificada neste nível.

O indivíduo, que manifesta comportamento, neste nível, é claramente percebido como adotando o valor. Age para favorecer a coisa valorizada de alguma maneira, para estender a

possibilidade de desenvolvê-la, para aprofundar seu envolvimento com ela e com as coisas que a representam. Tenta convencer os outros e procura convertidos à sua causa. Há uma tensão aqui, que precisa ser satisfeita; ação é o resultado de uma necessidade ou de impulsos surgidos. Existe uma motivação real para colocar o comportamento em ação. São exemplos: devoção à lógica e defesa à idéia de que ela é necessária para que o aluno se torne bom profissional em computação; devoção ao método de Programação em Duplas; entre outros.

2.3.4 Organização

À medida que o aprendiz internaliza sucessivamente valores, encontra situações, em relação às quais mais de um valor é pertinente. Desta maneira, surge a necessidade: (1) da organização de valores num sistema; (2) da determinação das inter-relações entre eles; e (3) do estabelecimento de valores dominantes e universais. Tal sistema é construído gradualmente, sujeito à mudança, à medida que novos valores são incorporados. Esta categoria está relacionada aos objetivos que descrevem os inícios da construção de um sistema de valores. É subdividida em dois níveis, desde que um pré-requisito para a inter-relação é a conceitualização do valor em uma forma que permita organização. Conceitualização constitui a primeira subdivisão no processo de organização; Organização de um sistema de valores, a segunda.

2.3.4.1 Conceitualização de um Valor

Neste nível, o indivíduo percebe como os valores se relacionam àqueles que ele já adotou ou aos novos, que venha a adotar. A conceitualização é abstrata e, neste sentido, é simbólica, mas os símbolos não precisam ser símbolos verbais. São exemplos: o aprendiz tenta identificar as características de um algoritmo que admira ou considera um dos melhores para o seu objetivo; o aprendiz forma julgamentos a respeito da responsabilidade dos software na sociedade do conhecimento; entre outros.

2.3.4.2 Organização de um Sistema de Valores

Os objetivos apropriadamente classificados aqui são aqueles que requerem que o aprendiz reúna um complexo de valores e que neles introduza uma relação ordenada entre si. Idealmente, a relação ordenada será aquela que é harmoniosa e internamente consistente. Este é, por certo, o alvo desses objetivos, que procuram fazer o estudante formular uma filosofia de vida. A integração às vezes pode ser menos do que inteiramente harmoniosa. Mais provavelmente, a relação é melhor descrita como um tipo de equilíbrio dinâmico, que está, em parte, dependente daquelas partes do ambiente que são notáveis em qualquer momento. Em muitos casos, a organização de valores pode resultar, em sua síntese, em um novo valor ou complexo de valores de uma ordem mais elevada. São exemplos: o aprendiz confronta orientações e práticas alternativas de trabalho colaborativo de preferência com os padrões de trabalho individual; o aprendiz desenvolve um plano de organização do seu tempo para resolver as atividades extraclasse e gerenciar seu tempo de descanso; entre outros.

2.3.5 Caracterização por um Valor ou Complexo de Valores

Neste nível de internalização, os valores já têm um lugar na hierarquia de valores do indivíduo, são organizados em algum tipo de sistema internamente consistente, controlaram o comportamento do indivíduo por um tempo suficiente, que ele se ajustou a se comportar desta maneira, e uma evocação do comportamento não mais suscita emoção ou afeto, exceto quando o indivíduo é ameaçado ou desafiado.

O indivíduo age consistentemente, de acordo com os valores que internalizou neste nível, e a preocupação aqui é indicar duas coisas: (1) a generalização deste controle para tanto do comportamento do indivíduo, que ele é descrito e caracterizado como uma pessoa, por estas tendências controladoras universais e (2) a integração dessas crenças, idéias e atitudes numa filosofia total ou numa visão do mundo. Estes dois aspectos constituem as subcategorias.

2.3.5.1 Direção Generalizada

A direção generalizada é o que dá uma consistência interna ao sistema de atitudes e valores, em qualquer momento específico. É uma resposta seletiva em um nível muito alto. Às vezes, é expressa como uma tendência determinante, uma orientação em relação a fenômenos, ou uma predisposição para agir de uma certa maneira. A direção generalizada é uma resposta a fenômenos altamente generalizados. É uma resposta persistente e consistente a um grupo de situações ou objetos relacionados. Muitas vezes, pode ser uma predisposição inconsciente, que guia a ação sem premonição consciente. A direção generalizada pode ser julgada como intimamente relacionada à idéia de um conjunto de atitudes, em que o que os membros integrantes têm em comum é baseado antes em características comportamentais, do que no assunto ou objeto da atitude. Uma direção generalizada é uma orientação básica, que capacita o indivíduo a reduzir e ordenar o mundo complexo à sua volta e agir consistente e eficientemente no mesmo. São exemplos: prontidão para revisar julgamentos e modificar trechos de seu próprio código fonte à luz de evidência; o aprendiz julga problemas e questões em termos de situações, controvérsias, propósitos, consequências subentendidas, mais do que em termos de preceitos fixos, dogmáticos ou pensamento emocionalmente ansioso; entre outros.

2.3.5.2 Caracterização

Este, o ponto culminante do processo de internalização, inclui aqueles objetivos que são mais amplos no que diz respeito tanto aos fenômenos abrangidos, quanto ao alcance do comportamento que eles compreendem. Desta maneira, aqui são encontrados aqueles objetivos, que dizem respeito à visão que a pessoa tem do universo, à sua filosofia de vida, ao seu sistema de valores que tem, como objeto, o conjunto do que é conhecido ou conhecível.

Segundo Bloom e colaboradores (1964), os objetivos categorizados nesse nível são mais do que direções generalizadas, no sentido de que envolvem uma ênfase na consistência interna, dentro do grupo de atitudes, comportamentos, crenças ou idéias. Ainda que esta

consistência interna nem sempre possa ser manifestada comportamentalmente, pelos estudantes, em relação a quem o objetivo se dirige, este traço de consistência será sempre um componente dos objetivos de Caracterização.

Como o título desta categoria subentende, estes objetivos são tão inclusivos que tendem a caracterizar o indivíduo quase completamente. São exemplos: o aprendiz desenvolve um código de comportamento para desenvolver algoritmos de maneira eficiente e eficaz; o aprendiz desenvolve um padrão de codificação relacionado aos critérios de qualidade de software; entre outros.

2.4 O RELACIONAMENTO ENTRE OS DOMÍNIOS COGNITIVO E AFETIVO

Algumas das mais interessantes relações entre os domínios cognitivo e afetivo (e algumas das indicações mais claras das inter-relações entre os dois domínios) são aquelas em que a consecução de um alvo ou objetivo de um domínio é considerada como o meio para a consecução de um alvo ou objetivo, no outro. Em alguns casos, pode-se utilizar modificações no domínio cognitivo, como um meio para produzir modificações no domínio afetivo; por exemplo, dá-se ao estudante informações com a intenção de mudar a sua atitude. Em outros casos, pode-se usar um alvo afetivo como um meio de atingir um cognitivo; por exemplo, desenvolve-se um interesse no material, de maneira que o estudante aprenderá a usá-lo (BLOOM e colaboradores, 1964).

2.4.1 Objetivos Cognitivos como Meio para Metas Afetivas

O observador cuidadoso, na sala de aula, pode perceber que o professor sagaz, tanto quanto o teórico psicológico utiliza o comportamento cognitivo e a consecução de alvos cognitivos, para atingir alvos afetivos (Figura 2.2). Em muitos casos, age assim, mais

intuitivamente do que conscientemente. Grande parte do que é chamado de “ensino satisfatório” é a capacidade do professor de alcançar objetivos afetivos, através do desafio a crenças fixadas dos estudantes e levando-os a discutirem questões (BLOOM e colaboradores, 1964).

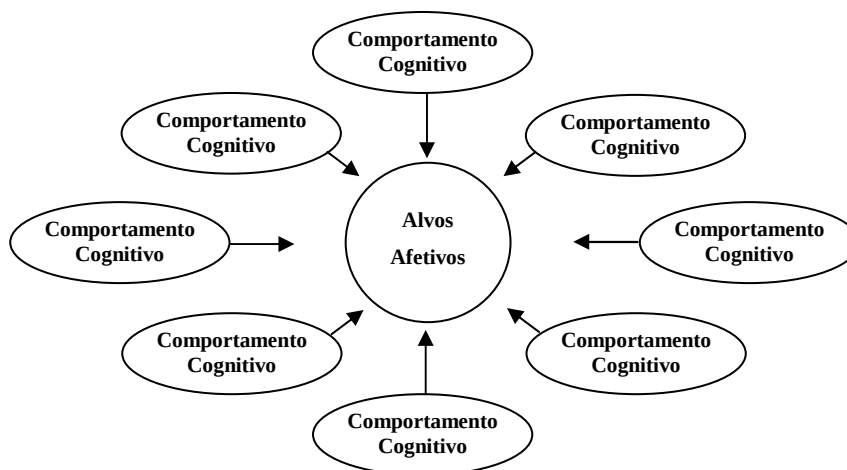


Figura 2.2 – Comportamento Cognitivo para Atingir Alvos Afetivos.

Em alguns casos, os professores utilizam o comportamento cognitivo, não apenas como um meio para o comportamento afetivo, mas como um tipo de pré-requisito. Os objetivos de apreciação são frequentemente abordados cognitivamente, fazendo, por exemplo, o estudante analisar um algoritmo, de maneira que chegará a compreender o modo pelo qual certas operações são realizadas – a combinação de instruções, a lógica, o estilo de programação, etc. Tal análise, num nível cognitivo, quando conhecida a fundo, pode ser percebida como a aprendizagem necessária para a “verdadeira” apreciação de um algoritmo, ou para o julgamento de sua qualidade sob uma visão ampla.

2.4.2 Objetivos Afetivos como Meio para Metas Cognitivas

Pela discussão prévia, parece claro que a abordagem cognitiva, para objetivos afetivos, é frequentemente uma rota percorrida. E o caminho inverso? Um dos principais tipos de objetivos do domínio afetivo, que são procurados como meios para os fins cognitivos, é o

desenvolvimento do interesse ou motivação. Na medida em que é encarado pelo pólo cognitivo, o estudante pode ser tratado como uma máquina analítica, como um “computador” que resolve problemas. Pelo contrário, considerado pelo pólo afetivo, toma-se maior conhecimento da motivação, impulsos e emoções, que são os fatores que levam à consecução do comportamento cognitivo.

De forma óbvia, a motivação é crítica para a aprendizagem e, desta maneira, é um dos modos principais, pelos quais o domínio afetivo é usado como meio para o cognitivo (Figura 2.3). O grande número de objetivos de interesse indica a importância deste aspecto da situação de aprendizagem. A influência da emoção sobre a memória e a aprendizagem também é importante: segundo Bloom e colaboradores (1964), indivíduos têm mais probabilidade de aprender e recordar material, em relação ao qual têm um sentimento positivo. São exemplos desse fato: (1) um aluno que não gosta de matemática tem muita dificuldade em aprendê-la; (2) meninos desgostosos de ir à escola dificilmente atingem resultados em avaliações melhores que aqueles que a apreciam. Ainda que estes “gostos” possam ser produzidos pelas expectativas, são as preferências internalizadas que produzem o efeito.

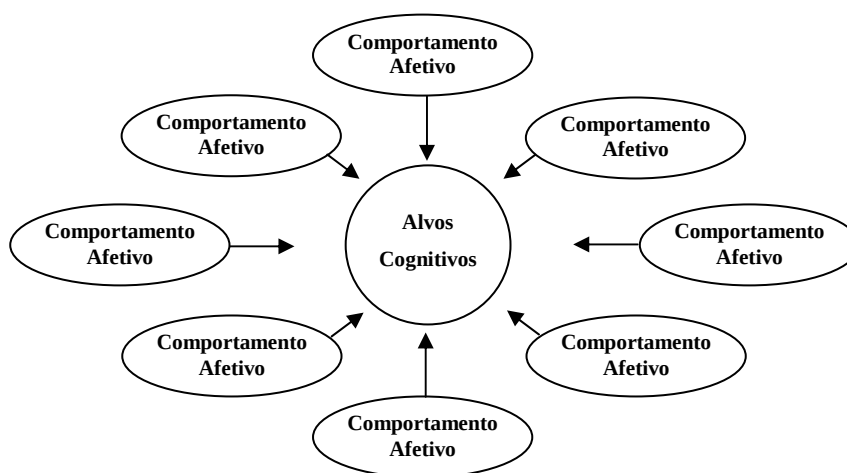


Figura 2.3 – Comportamento Afetivo para Atingir Alvos Cognitivos.

Onde estão envolvidos objetivos educacionais, quase sempre se tem interesse no afeto positivo, mais no mostrar o caminho, do que no impelir o aluno para a aprendizagem. Mas existem algumas situações escolares em que o efeito negativo é usado para impedir que ocorram certos comportamentos e para facilitar a aprendizagem cognitiva. Em alguns casos,

pode ser exercida a pressão social para modificar uma posição ou ponto de vista de um estudante.

Porém, a literatura teórica e experimental sugere que este não é um caminho fácil para a modificação cognitiva. Existe também a probabilidade de que as pessoas possam se submeter exteriormente, sob tais situações, mas permanecer sem modificação, interiormente. A teoria de Festinger (1957, apud BLOOM e colaboradores, 1964) da dissonância cognitiva postula que a ameaça ou pressão externas severas representam uma justificação, para o indivíduo se entregar a um comportamento, contrário a suas crenças, de maneira que há menos necessidade de reduzir a dissonância, causada por se envolver neste comportamento, sob as condições de ameaça. Onde a ameaça é leve, há menos justificação para se entregar ao comportamento e pode-se assim esperar mais modificação na opinião particular, para reduzir a dissonância. Por outro lado, certa atmosfera escolar ameaçadora poderia na verdade invalidar as tentativas dos professores de ocasionar tanto aprendizagem cognitiva como afetiva.

2.4.3 Concluindo Sobre a Teoria do Relacionamento Entre os Dois Domínios

O fato que deve ficar claro é que os dois domínios são estreitamente entrelaçados. Cada comportamento afetivo tem um equivalente de comportamento cognitivo de algum tipo e vice-versa. Um objetivo num domínio tem um equivalente no domínio oposto, ainda que frequentemente não se tome conhecimento disso. Existe uma certa correlação entre os níveis da Taxionomia, de um objetivo afetivo e seu equivalente cognitivo. Cada domínio é algumas vezes usado como um meio para o outro, ainda que a rota mais comum seja do cognitivo para o afetivo. Existem exposições teóricas que permitem expressar uma em termos da outra e vice-versa.

2.5 A TAXIONOMIA DE BLOOM REVISADA POR ANDERSON E COLABORADORES

Anderson e colaboradores (2001) publicaram uma versão atualizada da taxionomia de Bloom que considera, segundo os autores, uma gama maior de fatores que afetam o ensino e a aprendizagem sob o domínio cognitivo. A taxionomia revisada diferencia “saber o quê” (o conteúdo do raciocínio) de “saber como” (os procedimentos para resolver problemas).

A Dimensão do Conhecimento se divide em quatro categorias: factual, conceitual, procedimental e metacognitiva. O conhecimento factual inclui elementos isolados de informação, como definições de vocabulário e conhecimento de detalhes específicos. O conhecimento conceitual consiste em sistemas de informação, como classificações e categorias. O conhecimento procedimental (saber como fazer) inclui algoritmos, heurística ou método empírico, técnicas e métodos, bem como o conhecimento sobre quando usar esses procedimentos. O conhecimento metacognitivo (refletir sobre o que se sabe) refere-se ao conhecimento dos processos cognitivos e das informações sobre como manipular esses processos de forma eficaz.

O domínio cognitivo da taxionomia revisada de Bloom, como na versão original, é formado por seis categorias, da mais simples à mais complexa, são elas: Lembrar (Conhecimento), Entender (Compreensão), Aplicar (Aplicação), Analisar (Análise), Avaliar (Avaliação) e Criar (Síntese). Anderson e colaboradores (2001) renomearam a categoria “Síntese” como “Criar” e colocaram-na, na nova taxionomia, em uma ordem invertida com a categoria “Avaliação”.

De acordo com os autores, “Lembrar” consiste em reconhecer e recordar informações importantes da memória de longo prazo. “Entender” é a capacidade de fazer sua própria interpretação do material educacional, como leituras e explicações do professor. As ações desse processo incluem interpretação, exemplificação, classificação, resumo, conclusão, comparação e explanação. O terceiro processo, “Aplicar”, refere-se a usar o procedimento aprendido em uma situação familiar ou nova. O processo seguinte é “Analisar”, que consiste em dividir o conhecimento em partes e pensar como essas partes se relacionam com a estrutura geral. A análise é feita por meio de diferenciação, organização e atribuição. O

processo “Avaliar” engloba verificação e crítica. Por fim, “Criar” envolve reunir elementos para dar origem a algo novo. Para conseguir criar tarefas, os alunos geram, planejam e produzem.

2.5.1 A Dimensão do Conhecimento na Taxionomia Revisada

O Conhecimento **factual** corresponde a possuir informações básicas, como por exemplo, conhecimento de terminologia (termos do vocabulário de uma língua, símbolos matemáticos, símbolos de uma partitura musical, alfabeto) e conhecimento de detalhes e elementos específicos (componentes da pirâmide alimentar, cargos públicos eletivos em âmbito municipal, estadual e federal, principais batalhas na era romana).

O Conhecimento **conceitual** diz respeito às relações entre as partes de uma estrutura maior que as fazem funcionar em conjunto, como por exemplo, conhecimento de classificações e categorias (espécies de animais, tipos diferentes de argumentos, eras geológicas), conhecimento de princípios e generalizações (tipos de conflito na literatura, lei do movimento de Newton, princípios da democracia) e conhecimento de teorias, modelos e estruturas (teoria da evolução da espécie, teorias econômicas, modelos de DNA).

Conhecimento **procedimental** está relacionado a “como fazer algo”, como por exemplo, conhecimento de habilidades específicas do assunto e algoritmos (procedimento para resolver equações quadráticas, misturar cores para pintura a óleo, saque no vôlei), conhecimento de técnicas e métodos específicos do assunto (crítica literária, análise de documentos históricos, métodos para resolução de problemas matemáticos) e conhecimento dos critérios para determinar quando usar os procedimentos apropriados (métodos apropriados para tipos diferentes de experimentos, procedimentos de análise estatística usados em situações distintas, padrões para gêneros diferentes de escrita).

Conhecimento **metacognitivo** envolve “saber raciocinar de modo geral ou específico”, como por exemplo, conhecimento estratégico (maneiras de memorizar fatos, estratégias para compreensão da leitura, métodos para planejar um *site*), conhecimento sobre tarefas

cognitivas, incluindo o conhecimento contextual e condicional apropriado (conhecer as diferenças entre redigir e-mails e cartas comerciais) e autoconhecimento (necessidade de um diagrama ou gráfico para entender processos complexos, melhor compreensão em ambientes silenciosos, necessidade de discutir idéias com alguém antes de redigir um ensaio).

2.6 A TAXIONOMIA DE BLOOM CONDENSADA

A tabelas 2.1 e 2.2 ilustram a taxionomia de Bloom de uma forma condensada. A primeira coluna da tabela 2.1 mostra os objetivos de aprendizagem descritos por Bloom para o domínio cognitivo. A segunda coluna indica os processos para atingi-los e a terceira descreve os produtos resultantes da aprendizagem.

Tabela 2.1 – Objetivos da Taxionomia de Bloom e Processos para Atingi-los – Domínio Cognitivo

Objetivos	Processos ou Verbos de Ação Associados	Produtos Resultantes
Conhecimento - Especificação - Modos e meios para lidar com itens específicos - Fatos universais e abstrações em um dado campo	Definir, Reconhecer, Recitar, Identificar, Rotular, Compreender, Examinar, Mostrar, Coletar, Listar.	Rótulos, Datas, Nomes, Fatos, Definições, Conceitos.
Compreensão - Tradução - Interpretação - Extrapolação	Traduzir, Interpretar, Explicar, Descrever, Resumir, Demonstrar.	Argumento, Explicação, Descrição, Resumo.
Aplicação Uso de abstrações em situações específicas e concretas	Aplicar, Solucionar, Experimentar, Demonstrar, Construir, Mostrar, Fazer, Ilustrar, Registrar.	Diagrama, Ilustração, Coleção, Mapa, Jogo ou Quebra-Cabeças, Modelo, Relato, Fotografia, Lição.
Análise - Elementos - Relacionamentos - Princípios organizacionais	Conectar, Relacionar, Diferenciar, Classificar, Arranjar ou Estruturar, Agrupar, Interpretar, Organizar, Categorizar, Retirar, Comparar, Dissecar, Investigar.	Gráfico, Questionário, Categoria, Levantamento, Tabela, Delineamento, Diagrama, Conclusão, Lista, Plano, Resumo.
Síntese - Comunicação inédita - Plano de operação - Conjunto de relacionamentos abstratos	Projetar, Reprojetar, Combinar, Consolidar, Agregar, Compôr, Formular Hipótese, Construir, Traduzir, Imaginar, Inventar, Criar, Inferir, Produzir, Predizer.	Poema, Projeto, Resumo de Projeto, Fórmula, Invenção, História, Solução, Máquina, Filme, Produto Manufaturado ou Software.

Tabela 2.1 – Objetivos da Taxionomia de Bloom e Processos para Atingi-los – Domínio Cognitivo

<<Continuação>>

Objetivos	Processos ou Verbos de Ação Associados	Produtos Resultantes
Avaliação • Julgamento em termos de evidência interna • Julgamento em termos de evidência externa	Interpretar, Verificar, Julgar, Criticar, Decidir, Discutir, Verificar, Disputar, Escolher.	Opinião, Julgamento, Recomendação, Veredicto, Conclusão, Avaliação, Investigação, Editorial.

A primeira coluna da tabela 2.2 mostra os objetivos de aprendizagem descritos por Bloom para o domínio afetivo e a segunda coluna indica o conjunto de verbos de ação associados.

Tabela 2.2 – Objetivos da Taxionomia de Bloom e Processos para Atingi-los – Domínio Afetivo

Objetivos	Processos ou Verbos de Ação Associados
Acolhimento (atenção) • Percepção • Disposição para Receber • Atenção Controlada ou Seletiva	Ouvir, Aceitar, Estar Consciente, Atender, Receber, Favorecer, Preferir, Perceber, Selecionar.
Resposta • Aquiescência na Resposta • Disposição para Responder • Satisfação na Resposta	Especificar, Selecionar, Gravar, Responder, Listar, Desenvolver, Completar, Escrever, Derivar.
Valorização • Aceitação de um Valor • Preferência por um Valor • Cometimento	Aceitar, Incrementar, Indicar, Reconhecer, Desenvolver, Decidir, Participar, Realizar, Influenciar.
Organização • Conceitualização de um Valor • Organização de um Sistema de Valores	Organizar, Encontrar, Associar, Julgar, Determinar, Formar, Relacionar, Correlacionar, Selecionar.
Caracterização por um Valor ou Complexo de Valores • Direção Generalizada • Caracterização	Revisar, Aceitar, Demonstrar, Modificar, Julgar, Identificar, Enfrentar, Desenvolver, Decidir.

É importante destacar que os processos ou verbos de ação descritos nas duas tabelas anteriores devem ser utilizados para auxiliar na produção de materiais avaliativos em relação a cada um dos objetivos a ser alcançado, seja no domínio cognitivo ou no afetivo.

2.7 CONCLUINDO SOBRE A TAXIONOMIA DE OBJETIVOS EDUCACIONAIS DE BLOOM

Embora os estudos de Bloom e sua equipe tenham se tornado referência em termos de objetivos educacionais e avaliação baseada em objetivos, acredita-se que muitos educadores negligenciam o seu uso. Na esfera do ensino superior a situação não é diferente, pois muitos professores não estão atentos à existência de tecnologias pedagógicas deste porte. Isso ocorre porque o ensino superior (em geral) exige dos seus candidatos a professores, não conhecimentos pedagógicos, mas conhecimentos de metodologia do trabalho científico (amplamente difundidos nos programas de pós-graduação *strictu-sensu*).

CAPÍTULO 3

PROGRAMAÇÃO EM DUPLAS (*PAIR PROGRAMMING* – PP)

Programação em Duplas (PP) é uma prática colaborativa de desenvolvimento de software – que foi adicionada à Programação Extrema (*eXtreme Programming*¹ – XP) como uma de suas 12 práticas chave – em que dois programadores trabalham ao mesmo tempo em um único computador e na mesma tarefa de programação. Um dos programadores é comumente chamado de piloto (*driver*). O piloto controla o teclado e o mouse e ativamente executa a tarefa de digitação do programa. O outro, conhecido como navegador (*navigator*), assiste o trabalho do piloto e age como um sócio em busca de geração de idéias. O navegador, constantemente, revisa os dados digitados para identificar deficiências, enquanto procura erros de sintaxe e de lógica e implementações que desrespeitem as normas por eles impostas durante o projeto. Depois de um período de tempo pré-determinado, os sócios invertem seus papéis.

Os papéis assumidos entre navegador e piloto são complementares um ao outro e por diversas vezes se misturam durante o processo de desenvolvimento, ou seja, em alguns momentos, é possível encontrar os dois sócios executando mentalmente um único papel – o de navegador ou de piloto –, porém, é incentivado que a atividade de digitação do código fonte a ser produzido seja realizada por apenas uma pessoa de cada vez – aquele que se encontrar na posição de piloto naquele instante.

¹ Programação Extrema é uma abordagem de desenvolvimento ágil de software que credita parte do seu sucesso ao uso de Programação em Duplas por todos os programadores. A dimensão de PP em XP requer que dois programadores trabalhem simultaneamente não somente na codificação, mas também no projeto, algoritmos, e testes (CIOLKOWSKI e SCHLEMMER, 2002).

É importante ressaltar que PP é ideal para alunos ativos que aprendem por interação social (ZUALKERNAN, 2007) e profissionais que possuem habilidades para o trabalho colaborativo e facilidades em comunicação.

PP não é uma prática nova. Ainda em meados da década de 90, Constantine (1995) fez um dos primeiros relatos em que se observou o uso de emparelhamento de programadores no desenvolvimento de software (NAWROCKI e WOJCIECHOWSKI, 2001). Naquele mesmo ano, Coplien (1995) publicou seu livro sobre processo de produção de software sugerindo um padrão organizacional de desenvolvimento emparelhado.

Muito do interesse em PP tem provavelmente aumentado devido à introdução de Programação Extrema no mercado de desenvolvimento de software (CUBRANIC, STOREY e RYALL, 2006 e LEJEUNE, 2003). Neste sentido, essa metodologia trouxe uma atenção considerável à programação colaborativa.

Esses relatos, em conjunto com a introdução de PP na indústria de software pela metodologia XP, suscitaram a curiosidade dos especialistas em desenvolvimento de software. Motivado por esta curiosidade, Nosek (1998) publicou um experimento sobre PP e concluiu que tal prática aumentou em 100% o desempenho dos programadores, além de tornar o processo de solução de problemas mais prazeroso. Não obstante, Williams e colaboradores (2000), após terem aplicado um experimento estruturado em uma classe de estudantes de Engenharia de Software, na Universidade de Utah, confirmaram os relatos de Nosek e mostraram que o desenvolvimento de software com o uso de PP resulta em um produto mais confiável devido à quantidade reduzida de erros (*bugs*).

Desde então, diversos estudos têm procurado verificar os benefícios de PP no processo de desenvolvimento de software e no ensino-aprendizagem de técnicas de programação em cursos relacionados à computação.

A seguir são descritos os diversos benefícios do uso de PP reivindicados na literatura. Enquanto alguns benefícios foram mais questionados que outros, um conjunto deles ocorreu de forma unânime entre os pesquisadores. Dentre os benefícios unânicos, se destacam:

1. Melhora a qualidade global do software (HANKS, 2002, MCDOWELL, HANKS e WERNER, 2003, WILLIAMS e KESSLER, 2000, WILLIAMS e KESSLER, 2001,

- MCDOWELL e colaboradores, 2006);
2. Promove a transferência de informação e conhecimento entre os membros de equipes (CHONG e HURLBUTT, 2007, VANHANEN, 2007);
 3. Aumenta o moral e a confiança dos programadores na própria solução (HANKS, 2002, WILLIAMS e colaboradores, 2000, ZIN, IDRIS e SUBRAMANIAM, 2006);
 4. Aumenta a satisfação dos desenvolvedores durante as atividades de programação (CLIBURN, 2003, HANKS, 2002, HANKS e colaboradores, 2004, WILLIAMS e colaboradores, 2000, GEHRINGER e colaboradores, 2006);
 5. Reduz a taxa de erros encontrados nos testes de unidade devido às revisões e testes contínuos durante o desenvolvimento (DECLUE, 2003, PADBERG e MÜLLER, 2003, ARISHOLM e colaboradores, 2007, VANHANEN, 2007);
 6. Aumenta a responsabilidade dos programadores devido à pressão da dupla (HO e colaboradores, 2004, WILLIAMS e KESSLER, 2001);
 7. Aprimora as habilidades colaborativas dos programadores (WILLIAMS e KESSLER, 2000, PRESTON, 2006, WILLIAMS, 2006);
 8. Aprimora as habilidades de comunicação dos programadores (WILLIAMS, 2006);
 9. Diminui a carga de trabalho dos instrutores quando usada academicamente (CLIBURN, 2003, NAGAPPAN e colaboradores, 2003, WILLIAMS e KESSLER, 2001, WILLIAMS e colaboradores, 2002);
 10. Diminui a taxa de evasão de alunos nos cursos (MCDOWELL, 2006, MCDOWELL e colaboradores, 2006, NAGAPPAN e colaboradores, 2003, WILLIAMS e colaboradores, 2002);
 11. Conduz os alunos para uma atitude mais positiva em relação ao curso e à informática (WILLIAMS e colaboradores, 2003, WILLIAMS e colaboradores, 2002, SHERRELL e ROBERTSON, 2006);
 12. Aumenta o interesse dos alunos em se especializar em computação (WILLIAMS e colaboradores, 2003, MCDOWELL e colaboradores, 2006, WILLIAMS, 2006);
 13. Provê aos programadores, ao observarem seus sócios, o conhecimento de diferentes abordagens para a solução de problemas (CHONG e HURLBUTT, 2007, VANHANEN, 2007);
 14. Resulta em código menor (VANHANEN, 2007, SHERRELL e ROBERTSON, 2006).

Não obstante, os benefícios a seguir foram relatados por alguns autores e questionados por outros:

15. Reduz o tempo de desenvolvimento do software (DECLUE, 2003, PADBERG e MÜLLER, 2003, WILLIAMS e KESSLER, 2001, MATZKO e DAVIS, 2006, ARISHOLM e colaboradores, 2007);
16. Reduz o custo de desenvolvimento (NOSEK, 1998, SHERRELL e ROBERTSON, 2006);
17. Produz código mais eficiente (WILLIAMS e colaboradores, 2000, MATZKO e DAVIS, 2006, MCDOWELL e colaboradores, 2006);
18. Aumenta o sucesso dos alunos em exames avaliativos (MCDOWELL, HANKS e WERNER, 2003, WILLIAMS e colaboradores, 2002, MENDES, AL-FAKHRI e LUXTON-REILLY, 2006);
19. Aumenta o sucesso dos alunos em disciplinas de programação futuras (WILLIAMS e colaboradores, 2003);
20. Reduz a taxa de erros encontrados nos testes de integração devido às revisões e testes contínuos durante o desenvolvimento (DECLUE, 2003, PADBERG e MÜLLER, 2003, ARISHOLM e colaboradores, 2007, VANHANEN, 2007).

O grande consenso entre os estudos envolve o uso de PP como prática chave no ensino de técnicas de programação. Até mesmo os autores de trabalhos classificados como mercadológicos (pesquisas realizadas com desenvolvedores profissionais) acreditam nos benefícios que PP agrega ao processo de ensino-aprendizagem.

3.1 TEMAS CHAVE EM PROGRAMAÇÃO EM DUPLAS

Durante o levantamento bibliográfico sobre PP e após leitura e análise cuidadosa do mesmo, procurou-se identificar os temas chave de maior relevância sobre o assunto, dentre eles: 1) custo/benefício de PP; 2) PP na indústria de software versus PP na academia; 3) metodologias para o uso de PP no ensino de programação e na indústria de software; 4) diretrizes para o bom funcionamento de PP; 5) heterogeneidade versus homogeneidade em PP; 6) o impacto de PP aplicado a grupos minoritários e às mulheres; e 7) a pressão dos pares decorrente de PP.

3.1.1 Custo/Benefício de PP

Dois estudos específicos sobre análise de custo/benefício de PP foram encontrados na literatura. O primeiro, realizado por Cockburn e Williams (2001), relatou dois custos iminentes: (1) Custo de desenvolvimento; e (2) Custo de treinamento dos membros das equipes em habilidades colaborativas e em PP. Segundo os autores, 8 (oito) caminhos investigativos importantes justificam o uso de PP na indústria e na academia:

1. **Economia** – Apesar de exigir uma quantidade dobrada de recursos humanos em um período de tempo, na maioria das vezes, superior a 50% do tempo gasto por programadores individuais, o código resultante de emparelhamentos apresenta menor quantidade de erros. A economia com a remoção de erros supera em valor o aumento do custo de desenvolvimento;
2. **Satisfação** – Membros de equipes emparelhadas defendem PP como uma experiência mais agradável que o trabalho individual;
3. **Qualidade de projeto** – Pares produzem programas mais curtos, em muitos casos, indicando projetos superiores;
4. **Revisões contínuas** – Emparelhamento provê projeto e revisão ininterruptos de código, enquanto conduz, na maioria dos casos, a taxas eficientes de remoção de erros;
5. **Resolução de problemas** – Devido às habilidades colaborativas, pares tendem a resolver problemas “complexos” mais rapidamente;
6. **Aprendizado** – Emparelhamento permite que programadores aprendam uns com os outros;
7. **Construção de senso de equipe e melhoria nas habilidades de comunicação** – Os participantes de equipes emparelhadas aprendem a discutir e trabalhar juntos. Isto melhora e torna mais efetiva a comunicação entre os parceiros;
8. **Melhoria na administração dos projetos e redução de riscos** – Considerando a familiaridade dos participantes com as diversas partes do código, Programação em Duplas reduz o risco e o impacto da perda de pessoal.

Baseando-se em questionários aplicados aos participantes de experimentos empíricos prévios e em dados encontrados na literatura sobre os 8 caminhos investigativos, os autores se

apresentaram muito positivos e defenderam que os benefícios superam os seus custos de maneira significativa.

O segundo estudo foi realizado por Padberg e Muller (2003). De acordo com os autores, as vantagens da velocidade de programação e redução de defeitos são o grande potencial da PP. A pergunta é se o custo extra de PP é equilibrado pelos benefícios potenciais.

As vantagens do aumento da velocidade de programação e da redução de defeitos proporcionados pela PP normalmente são explicadas como se segue:

A Programação em Duplas permite que os desenvolvedores compartilhem suas idéias imediatamente. Isto colabora para que sejam encontradas soluções mais rapidamente e também ajuda a eliminar defeitos precocemente. Além disso, leva a uma revisão contínua do código do programa, reduzindo a sua densidade de erros (PADBERG e MULLER, 2003).

Os autores mostraram como combinar diferentes métricas para avaliar o custo e benefício de PP. Foram integradas métricas de processos, métricas de produtos e métricas de contexto de processos dentro de um modelo de valor de negócio de um projeto – o modelo é baseado em uma concepção de valor líquido atual, o qual permite considerar o impacto do tempo no valor de mercado de um projeto. O modelo considera ainda que o número de pares, a vantagem da velocidade de desenvolvimento e a redução de defeitos proporcionados pelo emparelhamento têm um forte impacto no valor de projetos que usam PP. Os resultados do estudo proporcionam um gerenciamento-guia esclarecendo quando PP é mais indicado e quando não é aconselhável.

A conclusão do estudo é a que se segue: (1) gerentes devem usar PP quando a pressão pela disponibilização do programa no mercado for muito forte, pois seus programadores são mais rápidos e produzem software de qualidade superior quando trabalham em pares; em comparação a quando trabalham individualmente – nestes casos, um curto tempo de negócio é decisivo para o sucesso do projeto; (2) por outro lado, se a quantidade de mão-de-obra disponível não permitir trabalhar com pares suficientes para explorar o grau de paralelismo possível no projeto, o gerente deve considerar a adição de mão-de-obra de programadores individuais, em vez de usar a PP.

3.1.2 PP na Indústria de Software Versus PP na Academia

Compreender as diferenças entre PP na academia e no mercado é primordial para pesquisadores na escolha de trabalhos futuros que envolvam Programação em Duplas e aprendizado colaborativo de programação. PP na academia se distingue de PP na indústria de software por diversos motivos. A tabela 4.1 distingue PP baseando-se nos Benefícios atribuídos a PP na seção 4.

Tabela 4.1 – Distinguindo PP na Indústria de Software de PP na Academia

Benefícios	PP na indústria de software	PP na Academia
Melhora a qualidade global do software	É um dos pilares que sustentam a viabilização de PP na indústria de software	Apresenta-se como consequência do aprendizado promovido pelo conflito e pela colaboração
Promove a transferência de informação e conhecimento entre os membros de equipes	Muito importante para tornar todos os membros de equipes em especialistas nas diversas nuances de um sistema	Permite aos membros emparelhados o conhecimento de novas habilidades lógicas e de novas instruções em linguagens de programação
Aumenta o moral e a confiança dos programadores na própria solução	Gera software de maior qualidade	Ajuda a diminuir a evasão dos alunos nos cursos de computação
Aumenta a satisfação dos desenvolvedores durante as atividades de programação	Permite um nível maior de união entre os membros de equipes de desenvolvimento	Ajuda a diminuir a evasão dos alunos nos cursos de computação
Reduz a taxa de erros encontrados nos testes de unidade devido às revisões e testes contínuos durante o desenvolvimento	Provê o desenvolvimento ágil de software de qualidade	Torna menos necessário o processo de depuração, muitas vezes dispendioso para alunos pouco experientes e impacientes
Aumenta a responsabilidade dos programadores devido à pressão da dupla (pressão dos pares)	Diminui a possibilidade dos membros da dupla se dispersarem – a dispersão, certamente, reduziria a produtividade	Diminui a possibilidade de plágio dos trabalhos realizados
Aprimora as habilidades colaborativas dos programadores	Importante para profissionais que pretendem trabalhar em grandes projetos de software	Prepara melhor o aluno para exercer carreiras em informática
Aprimora as habilidades de comunicação dos programadores	Importante para tornar os profissionais mais respeitados, aumentando a empregabilidade	Prepara melhor o aluno para exercer carreiras em informática
Diminui a carga de trabalho dos instrutores quando usado academicamente	Não se aplica	Permite que os instrutores ocupem seu tempo com questões e dúvidas mais complexas
Diminui a taxa de evasão de alunos nos cursos	Não se aplica	Aumenta a viabilidade de manutenção dos cursos de computação
Conduz os alunos para uma atitude mais positiva em relação ao curso e à informática	Não se aplica	Ajuda a diminuir a evasão dos alunos nos cursos de computação
Aumenta o interesse dos alunos em se especializar em computação	Não se aplica	Permite a expansão dos cursos de pós-graduação em computação

Tabela 4.1 – Distinguindo PP na indústria de software de PP na Academia
 <<Continuação>>

Benefícios	PP na indústria de software	PP na Academia
Provê aos programadores, ao observarem seus sócios, o conhecimento de diferentes abordagens para a solução de problemas	Torna os profissionais mais imprescindíveis em suas equipes	Aprimora as habilidades de resolução de problemas necessárias aos alunos
Resulta em código menor	Pode influenciar na eficiência e na eficácia dos softwares	Não se aplica
Reduz o tempo de desenvolvimento do software	Influencia diretamente no custo e na viabilidade dos softwares	Não se aplica
Reduz o custo de desenvolvimento	Influencia diretamente na viabilidade dos softwares	Não se aplica
Produce código mais eficiente	Potencializa a comercialização dos softwares	Não se aplica
Aumenta o sucesso dos alunos em exames avaliativos	Não se aplica	Ajuda a diminuir a evasão dos alunos nos cursos de computação
Aumenta o sucesso dos alunos em disciplinas de programação futuras	Não se aplica	Ajuda a diminuir a evasão dos alunos nos cursos de computação
Reduz as taxas de erros encontrados nos testes de integração devido às revisões e testes contínuos durante o desenvolvimento	Provê o desenvolvimento ágil de software de qualidade	Permite aos alunos a sensação de sucesso nos seus trabalhos

3.1.3 Metodologias Para o Uso de PP no Ensino de Programação e na Indústria de Software

De acordo com o protocolo de PP proposto pelos criadores da Programação Extrema, em um emparelhamento de programadores: dois desenvolvedores de software compartilham um único monitor e teclado – exercendo papéis de piloto e navegador; os programadores trocam seus papéis regularmente – a troca de papéis é um processo informal e um intervalo típico compreende 20 minutos; o rodízio dos pares é incentivado para promover a transmissão de informação e conhecimento entre os membros de equipes após execução de tarefas modulares.

Embora o protocolo direcione para um bom funcionamento de PP, diversas facetas podem tornar a metodologia de implantação de Programação em Duplas mais efetiva ou não, seja para fins acadêmicos, seja para fins industriais. Por exemplo, duplas podem ser

constituídas de maneira aleatória ou pré-definida; rodízio dos pares pode ser incentivado ou restringido; troca de papéis pode ser feita periodicamente ou quando os participantes assim desejarem.

Não foram encontrados estudos definindo ou sugerindo metodologias para implantação de PP. Ao contrário, a maioria dos trabalhos explorados procurou descrever os riscos e benefícios do uso de rodízio dos pares, troca de papéis e escolha dos pares.

3.1.3.1 Rodízio dos pares

Enquanto alguns estudos sugeriram o rodízio como solução para os problemas de compatibilidade da dupla, outros propuseram seus benefícios em relação à troca de conhecimentos e às habilidades de comunicação.

O rodízio dos pares é importante para que alguns benefícios de PP sejam alcançados. Esse é o consenso entre os especialistas, visto que poucos trabalhos em que PP foi aplicada declararam não trabalhar com tal prática.

É importante também ressaltar: quando o objetivo do emparelhamento estiver relacionado ao compartilhamento de informação e conhecimento, o rodízio dos pares é aparentemente eficiente e ideal.

3.1.3.2 Troca de papéis

De acordo com a literatura, a troca de papéis é importante para manter o equilíbrio técnico e emocional entre os membros da dupla. Na teoria dos autores, parece ser intuitivo que a troca de papéis impede os sócios de permanecerem em atitudes passivas – é defendido que o simples ato de digitar traz o sócio para a realidade ativa do desenvolvimento emparelhado.

Uma visão surpreendente e interessante foi encontrada no estudo de Chong e Hurlbutt (2007). Segundo os autores, além da tarefa de digitar, parece não haver uma divisão consistente de trabalho entre o piloto e o navegador. Ao invés disso, os dois programadores movem-se de tarefa a tarefa juntos, considerando e discutindo problemas no mesmo nível em termos de estratégia ou abstração. Assim, a passividade não estaria diretamente relacionada ao simples ato de dominar ou não o teclado, mas às atitudes dos sócios quando se comunicam e quando há conflito.

3.1.3.3 Escolha dos pares

Alguns estudos investigaram formação aleatória das duplas. Em sua grande maioria, ficou clara a intenção de se beneficiar do rodízio dos pares, justificando o emparelhamento aleatório – é defendido que o rodízio com composição aleatória permite aos participantes uma troca maior de experiências emocionais e habilidades lógicas.

Outros estudos abordaram composição pré-definida dos pares. Enquanto alguns permitiram emparelhamento escolhido pelos participantes, outros propuseram emparelhamento por nível de habilidade, semelhança e diferença de personalidade, autoconfiança semelhante, etnicidade semelhante, gênero semelhante, amor-próprio semelhante e por afinidade.

3.1.4 Diretrizes Para o Bom Funcionamento de PP

À medida que as pesquisas evoluíram no sentido de tornar a PP e a programação colaborativa mais explorada e aprimorada, alguns estudiosos estabeleceram, isoladamente, diretrizes de como aplicá-las de maneira mais efetiva. As diretrizes foram condensadas em um guia prático que direciona para boas atitudes dos aprendizes e professores em seções de Programação em Duplas (Apêndice C).

3.1.5 Heterogeneidade Versus Homogeneidade em PP

Estudos sobre composição de pares homogêneos e heterogêneos foram encontrados – a maioria empregou pesquisa empírica no intuito de verificar a efetividade de emparelhamentos pré-definidos heterogêneos e homogêneos por nível de habilidade, personalidade, autoconfiança, etnicidade, gênero e amor-próprio como meio de encontrar as variáveis que influenciam a compatibilidade dos pares.

Até mesmo um modelo de usuários foi desenvolvido em Yuksel (2005) com o propósito de verificar a compatibilidade dos pares. Os autores sugerem o desenvolvimento de sistemas inteligentes de emparelhamento harmonioso de sócios baseados no modelo.

Os autores que se mostram crédulos nos benefícios provindos de emparelhamentos homogêneos defendem a compatibilidade dos pares como primordial para o sucesso de práticas de PP, principalmente na indústria de software – mas isto é fácil de ser justificado. Intuitivamente, duplas heterogêneas aumentam o risco de sócios menos experientes se tornarem passivos, diminuindo sensivelmente os benefícios que PP agregaria ao projeto.

Os estudos de Mujeeb-U-Rehmann e colaboradores (2005), especificamente sobre heterogeneidade e homogeneidade das duplas, chegou a conclusões aparentemente contraditórias. Segundo os autores, em seus experimentos, as duplas heterogêneas provaram lidar melhor com a resolução dos problemas que inicialmente apareceram, e assim tiveram maior êxito na visualização de idéias bem como um potencial criativo mais apurado quando comparadas às duplas homogêneas. Em geral, as análises empíricas comparativas demonstraram que as duplas heterogêneas se saíram melhor do que as duplas homogêneas.

As conclusões dos autores parecem contraditórias às teorias em que se acredita que a heterogeneidade pode levar a um índice de passividade prejudicial ao processo de PP. Porém, no estudo supracitado, não fica claro se foram resolvidos e como foram resolvidos os problemas de dominância/complacência e passividade de sócios menos experientes. Concomitantemente, nada foi relatado a respeito de como os pares foram moderados, deixando, portanto, uma dúvida latente: “Duplas heterogêneas normalmente são compostas de membros muito experientes com membros iniciantes, ou, pelo menos, existe uma grande

possibilidade de que isto ocorra. Neste sentido, a causa do sucesso das duplas heterogêneas não poderia ser atribuída aos alunos mais experientes?” – Esta, certamente, é uma pergunta que ainda precisa ser respondida.

Em termos pedagógicos, ao utilizar PP como prática no apoio ao ensino-aprendizagem de programação de computadores, espera-se que o conhecimento seja disseminado através da colaboração e do conflito, em geral muito ocorrentes em duplas homogêneas. Por outro lado, em duplas heterogêneas, os sócios experientes, quando treinados para tal, podem aproveitar o processo de emparelhamento no intuito de transmitir seus conhecimentos e habilidades aos menos experientes. Neste sentido, talvez seja interessante que os instrutores pratiquem uma certa alternância de heterogeneidade e homogeneidade em emparelhamentos sucessivos, de modo que os aprendizes se beneficiem de ambas as experiências.

3.1.6 O impacto de PP aplicado a Grupos Minoritários e às Mulheres

Como é esperado da maioria das práticas pedagógicas, PP se mostra muito efetiva quando grupos minoritários formam seu público alvo. Diversos estudos procuraram medir a eficiência da prática de PP com grupos minoritários e com mulheres. Alguns resultados podem ser encontrados a seguir:

- Pares de gêneros diferentes são menos prováveis de serem compatíveis (KATIRA, OSBORNE e WILLIAMS, 2005);
- Pares de grupos minoritários são mais prováveis de serem compatíveis (KATIRA, OSBORNE e WILLIAMS, 2005);
- Ao programar com um sócio, mulheres acreditam que solucionaram problemas que não seriam capazes de resolver sozinhas – efeito causado pelo conflito (HO e colaboradores, 2004);
- PP influencia positivamente em menor evasão de alunos do sexo feminino (MCDOWELL e colaboradores, 2003, WERNER, DENNER e BEAN, 2004 e WERNER, HANKS e MCDOWELL, 2005);

- Existem fortes indícios de que PP é especialmente benéfica a estudantes do sexo feminino (BECK, CHIZHIK e MCELROY, 2005, HANKS, 2006 e WILLIAMS e colaboradores, 2007).

3.1.7 A Pressão dos Pares Decorrente de PP

Programadores emparelhados colocam uma forma positiva de pressão nos seus respectivos pares. Os programadores admitem que trabalham firme e bem mais atentos porque eles não gostam de deixar seus parceiros falharem. Também, quando se encontram em seções de emparelhamento, ambos trabalham intensivamente porque são altamente motivados a completar sua tarefa durante a sessão. Duas pessoas trabalhando de maneira emparelhada tornam seus respectivos tempos mais valiosos – eles tendem a deixar de fazer ligações curtas; eles não checam mensagens de e-mails ou páginas da web; eles não desperdiçam o tempo uns dos outros. À medida que cada um se concentra em sua atividade, ambos se mantêm focados nas tarefas e melhorias são realizadas (WILLIAMS, 2001).

Outro benefício da pressão dos pares é a adesão a procedimentos e padrões. Devido à natureza humana, os pares colocam uma forma positiva de pressão uns nos outros para seguir os processos orientados. A “pressão dos pares” faz os estudantes seguirem o processo e praticar o que o instrutor está ensinando de fato. Até mesmo quando eles têm vontade de saltar um passo importante do processo, como documentar o projeto, eles ficam envergonhados de relatar ao sócio ou o mesmo os persuade a completar todos os passos (WILLIAMS e KESSLER, 2001).

Williams (2001) e Williams e colaboradores (2006) relataram também que a “pressão dos pares” tende a motivar os estudantes a iniciar a solução das tarefas mais cedo.

É consenso que os pares de estudantes realmente aplicam uma forma positiva de “pressão dos pares” uns aos outros. Nos experimentos vistos no capítulo 7, esta pressão provou ser benéfica à qualidade dos produtos do trabalho deles, além de que as tarefas emparelhadas submetidas se apresentaram mais completas.

3.2 POR QUÊ USAR PP NO ENSINO DE PROGRAMAÇÃO DE COMPUTADORES?

Diversos são os motivos pelos quais optou-se em aplicar Programação em Duplas como auxílio ao ensino de “Algoritmos e Técnicas de Programação”, dentre eles:

- Acredita-se em metodologias pedagógicas alternativas quando são tratados problemas complexos e aparentemente sem solução – embora se acredita também não existir a solução perfeita para o problema do ensino-aprendizagem de programação de computadores, visto que diversas são as causas desse problema e que podem ocorrer concomitantemente;
- A literatura relata diversos benefícios que a PP pode agregar ao processo de ensino-aprendizagem de programação;
- O mercado de software, principalmente no momento atual (na era da globalização), tem envidado esforços no sentido de incentivar e privilegiar profissionais aptos para o trabalho em equipe. Acredita-se que essas habilidades devem ser adquiridas ainda nas disciplinas iniciais de cursos de computação;
- E finalmente, porque a programação solo não se mostra eficaz (e não se mostrou eficaz desde os primórdios da computação) para um percentual muito alto de aprendizes que falham nos seus objetivos de se tornarem programadores.

3.3 CONSIDERAÇÕES FINAIS IMPORTANTES PARA UMA IMPLANTAÇÃO BEM SUCEDIDA DE PP

Embora não tenha sido relatado nas seções anteriores, as observações abaixo foram consensuais na literatura sobre PP:

- É importante incentivar uma forte comunicação entre o navegador e o piloto para que ambos consigam acompanhar o raciocínio do parceiro;

- O acultramento dos participantes em atividades colaborativas é primordial para que metodologias de implantação de PP sejam bem sucedidas – as pessoas não estão acostumadas a colaborar, em vez disso, são altamente competitivas, motivo pelo qual PP pode falhar;
- Os participantes precisam ser treinados em PP para que seus benefícios sejam plenamente percebidos.

CAPÍTULO 4

TRABALHOS RELACIONADOS

Trabalhos correlatos foram encontrados na literatura. Enquanto alguns se preocuparam em aplicar taxionomias educacionais genéricas no ensino de programação (como, por exemplo, a taxionomia de Bloom), outros se ocuparam da construção de taxionomias especificamente projetadas para o manejo de conteúdos na área de computação. Os trabalhos seguintes se destacam.

4.1 ESQUEMA DE NIVELAMENTO DE ESTUDANTES BASEADO EM CRITÉRIOS E NA TAXIONOMIA DE BLOOM

Lister e Leaney (2003) descreveram um esquema de nivelamento baseado em critérios para estudantes do primeiro ano de programação que foi projetado para permitir que eles alcancem todo o seu potencial. Os autores acreditam que esquemas de nivelamento tradicionais, em que todos os estudantes trabalham na mesma tarefa avaliativa, resultam em tarefas que podem ser efetivas para o estudante de nível médio. Por outro lado, essas tarefas não permitem que os estudantes mais fracos se empenhem efetivamente e também não desafiam os estudantes mais fortes. O esquema de nivelamento proposto pelos autores usa uma mistura de várias estratégias de avaliação para superar esse problema. As estratégias foram usadas em ambientes de avaliação tradicionais, mas, separadamente para cada nível de estudante, considerando que as estratégias foram combinadas em uma filosofia de nivelamento coerente, explicitamente baseada na taxionomia de Bloom.

4.1.1 A Taxionomia de Bloom Aplicada aos Alunos de Acordo com Três Faixas de Aprovação

Ao definir seus critérios de nivelamento, Lister e Leaney (2003) aplicaram os seis níveis da taxionomia de Bloom (Conhecimento, Compreensão, Aplicação, Análise, Síntese e Avaliação) a três faixas de aprovação que são utilizadas na Universidade de Tecnologia de Sidney / Austrália, onde os estudos foram desenvolvidos. As faixas de aprovação são, respectivamente: (1) Aprovação Simples – P; (2) Crédito ou Distinção – C/D; (3) e Alta Distinção – HD.

Foi definido que:

- a) Uma pontuação “P” define um bom desempenho nos níveis de conhecimento e de compreensão;
- b) Uma pontuação “C/D” define um bom desempenho nos níveis de aplicação e de análise;
- c) Uma pontuação “HD” define um bom desempenho nos níveis de síntese e de avaliação.

Segundo os autores, a meta de “resolução de problemas” tradicional de um curso de programação introdutório – em que os estudantes deveriam, a partir de uma descrição de problema, ser capazes de escrever um programa bem estruturado e absolutamente correto – passa por todos os estágios da taxionomia de Bloom. Portanto, os critérios para um “P” e “C/D” não exigem que os estudantes estejam completamente aptos àquela meta tradicional; só os estudantes de pontuação “HD” estão aptos.

Foram criadas diversas tarefas a serem exigidas dos alunos. Essas tarefas foram classificadas de acordo com os seis níveis descritos nos objetivos educacionais de Bloom. Assim, exigiu-se dos estudantes a realização de tarefas baseadas nos níveis da taxionomia de Bloom única e exclusivamente de acordo com o nível de classificação de pontuação em que se encontrava cada aluno (Vide Figura 4.1) – alunos classificados em um semestre anterior (ou na primeira avaliação) com pontuação “P” foram considerados aptos para solucionar problemas do nível Compreensão e Conhecimento. Em seguida, exigiu-se deles a resolução

de problemas de Aplicação e Análise no intuito de se tornarem aptos a melhorar sua pontuação ao nível “C/D”. Alunos classificados com pontuação “C/D” foram considerados aptos para solucionar problemas do nível Aplicação e Análise e, portanto, foi-lhes exigida a solução de problemas de Síntese e Avaliação para se tornarem aptos a melhorar sua pontuação ao nível “HD”. Por fim, alunos classificados com pontuação “HD” foram submetidos a tarefas de nível de complexidade superior aos demais alunos. A complexidade das tarefas, exigidas dos alunos com pontuação “HD”, foi aumentada de tarefa a tarefa.

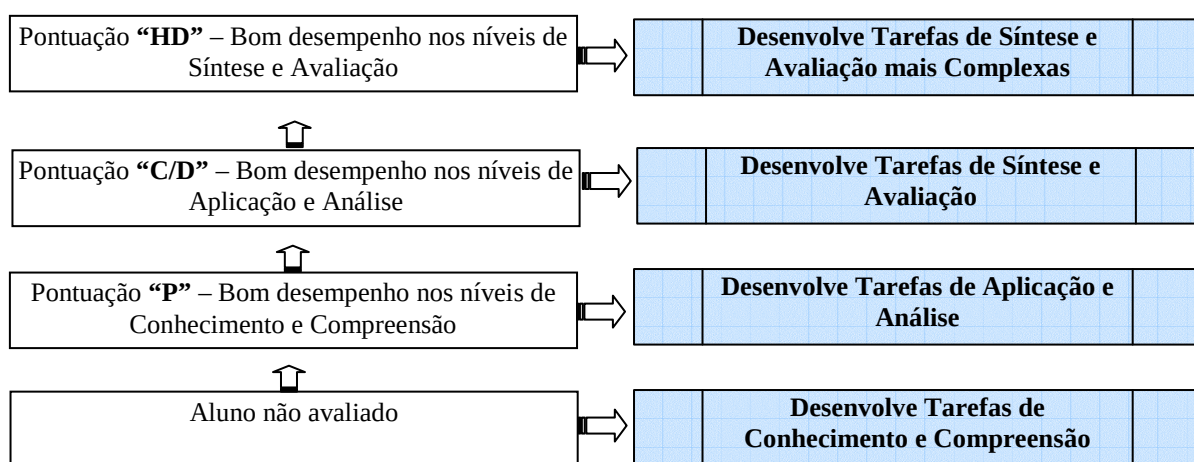


Figura 4.1 – Faixas de Aprovação e Esquema de Nivelamento de Acordo com as Categorias do Domínio Cognitivo de Bloom.

4.1.2 Concluindo Sobre o Esquema de Nivelamento de Estudantes de Lister e Leaney

O esquema de nivelamento pode ser questionado por não levar consideração os diversos fatores que influenciam os aprendizes a obterem pontuações consideradas altas, medianas ou baixas. Desta forma, fatores externos como, por exemplo, mal estar ou problemas emocionais durante uma avaliação, podem influenciar diretamente no nível em que será enquadrado um aprendiz, embora potencialmente ele possa ser considerado o melhor aluno de programação naquela classe. Deste ponto em diante, o processo de nivelamento falha nos seus objetivos, até que sejam aplicadas novas avaliações, pois o aprendiz estará

subestimado em relação às suas capacidades e habilidades, uma vez que lhe será solicitada a realização de tarefas em um nível inferior ao qual se encontra preparado.

4.2 A TAXIONOMIA DE TIPOS DE TAREFAS

Bower (2008) propôs uma taxionomia de tipos de tarefa que identificam os tipos diferentes de processos que os estudantes empreendem ao aprender programação de computadores. Segundo o autor, tipo de tarefa está definido como a atividade ou processo que é esperado que os estudantes executem para aprender os conceitos que lhes são apresentados. A característica definível da tarefa é que impõe o que os estudantes devem fazer de fato. Suas atividades podem variar desde recordar um fato a resolver um problema e a compreender a hierarquia de cada tarefa seguida. Para defender sua motivação, em relação a uma taxionomia educacional, Bower aponta para o fato de que os educadores em informática, até então, investiram muita energia definindo os conteúdos que se espera que os estudantes aprendam e se esqueceram do processo de ensino dos mesmos. Isto representa uma ênfase no conhecimento efetivo de conteúdos e não no processo de aprendizado.

4.2.1 As Tarefas

A taxionomia de tipos de tarefas proposta por Bower inclui 10 níveis de tarefas, dentre elas: (1) Tarefas declarativas; (2) Tarefas de compreensão; (3) Tarefas de depuração; (4) Tarefas de predição; (5) Tarefas de provimento de exemplo; (6) Tarefas de provimento de modelo; (7) Tarefas de Avaliação; (8) Tarefas de projeto; (9) Tarefas de resolução de problemas; e (10) Tarefas de reflexão.

4.2.1.1 Tarefas Declarativas

Conhecimento declarativo é estático e normalmente envolve a relação de mais de uma parte de informação. Este conhecimento trabalha ao nível de reconhecimento e lembrança. Tarefas declarativas são o mais baixo nível de tarefas que podem ser prescritas aos estudantes. É importante ressaltar que este conhecimento dá suporte a todas as tarefas de níveis mais altos. São exemplos de tarefas declarativas:

- Qual a sintaxe do laço “for”?
- Escreva a instrução de cálculo da raiz quadrada (contida na biblioteca “*math.h*”) de uma variável qualquer em C++;
- Escreva um fragmento de código utilizando a instrução “*switch-case*” para uma variável que poderá assumir valores entre 1 e 5 – contabilizando cinco situações de ações distintas.

4.2.1.2 Tarefas de Compreensão

Tipicamente, compreensão envolve a apresentação de um artefato (como um fragmento de código) ou um artigo de conhecimento declarativo e sua explicação (todo ou parte). São exemplos:

- Por que a sintaxe de um laço “for” possui um passo inicial, seguido de uma condição, seguida de um passo sequencial?
- Explique a diferença entre o inteiro 127 e a cadeia de caracteres “127”.

4.2.1.3 Tarefas de Depuração

Tarefas de depuração exigem que os estudantes descubram erros de programação no código, frequentemente fundadas em uma antecipação do que o programa está tentando alcançar. Tarefas de depuração sintáticas estão baseadas no conhecimento declarativo (sintaxe) enquanto tarefas de depuração semânticas se baseiam na compreensão da lógica empregada ao agrupar as instruções com uma determinada finalidade. São exemplos:

- Qual é o erro sintático encontrado no fragmento de código a seguir?

```
System.outprntln ("Oi);
```

- Qual é o erro semântico encontrado no fragmento de código a seguir?

```
int i;  
int fatorial = 1;  
for (i = 1; i < = 5; i++);  
    {fatorial = fatorial * i;}  
System.out.println ("Fatorial de " + i + " = " + fatorial);
```

4.2.1.4 Tarefas de Predição

Tarefas de predição são importantes na computação porque sem a habilidade para prever o efeito das instruções de um fragmento de código os estudantes não podem escrever programas. Predição correta está baseada na precisa compreensão e no conhecimento declarativo. Tarefas de predição são mais cognitivamente exigidas durante a tarefa de depuração. Assim, tarefas de predição exigem que os estudantes sejam generalistas e possam interpretar o código como um todo, além de identificar erros particulares. São exemplos:

- O que faz esta linha código “println” a seguir?

```
System.out.println ("11 + 5" + 20);
```

- Qual será o efeito de substituir os valores “i” por “j” no código seguinte?

```
void main() {  
    int i=0, j=1;  
    int vetor[10];  
    while(i<10) {  
        cin>>vetor[i];  
        i++;  
    }  
}
```

4.2.1.5 Tarefas de Provimento de Exemplo

Tarefas de provimento de exemplo são tarefas criativas que podem ser de natureza declarativa (efetivo e sintático) ou compreensiva (entendimento ou semântico). É importante ressaltar que estas tarefas exigem frequentemente um maior compromisso cognitivo que as tarefas anteriores. Isto ocorre porque os exemplos provêm pequenas sugestões das quais os estudantes podem recuperar o seu conhecimento (em um nível declarativo) ou exigem que os estudantes sintetizem fragmentos existentes de conhecimento para criar uma representação original. Esta é uma capacidade de pensamento de ordem mais alta e é uma das habilidades exigidas, por exemplo, na indústria de software. Veja os exemplos a seguir:

- Descreva um exemplo de um erro lógico;
- Construa um exemplo de código (condicional – “if”) em que o uso de “else” pode ser desnecessário. Reescreva o mesmo código sem o “else”.

4.2.1.6 Tarefas de Provimento de Modelo

Tarefas de provimento de modelo vão além de tarefas de provimento de exemplo para não só demonstrar a habilidade em entender exemplos que foram apresentados, mas também prover uma representação abstrata de uma situação completa ou um processo. Esta tarefa pode

ser classificada em um nível bastante baixo de demanda cognitiva se os estudantes representarem modelos que eles simplesmente encontraram em outro lugar. Por outro lado, se os estudantes são desafiados (ou se desafiam a si mesmos) a sintetizar seu conhecimento declarativo e de compreensão para derivar um modelo original, tais tarefas podem ser oportunidades ricas para relacionar e reestruturar conceitos, enquanto podem conduzir, assim, a uma compreensão mais profunda. São exemplos:

- Explique o que acontece quando seu computador executa um programa em Java (através da máquina virtual Java - JVM);
- Crie um diagrama para ilustrar o que acontece na memória do computador quando você: (1) cria uma variável objeto (ao definir um nome de variável novo e seu tipo); (2) inicializa aquela variável objeto.

4.2.1.7 Tarefas de Avaliação

Tarefas de avaliação previamente construídas são frequentemente associadas com a ordem mais alta do pensamento segundo a taxionomia de Bloom. A avaliação pode ser lançada de fato a uma grande variedade de níveis de dificuldade e pode alcançar muitos tipos diferentes de pensamento. Por exemplo, a tarefa “faça uma lista das vantagens de *applets* em aplicações” resultará frequentemente na reprodução de uma resposta de um livro texto ou uma informação encontrada na Internet. Porém, tarefas de avaliação também podem se aproximar de um nível especialista quando um aprendiz tentar decidir qual abordagem de processo de software é ideal a ser adotada durante a construção de um determinado sistema com suas diversas características (bem definidas e previamente conhecidas). Não obstante, avaliação pode acontecer durante as tarefas de provimento de exemplo e provimento de modelo.

Segundo Bower (2008), a tarefa de avaliação pode ser considerada uma boa opção para o uso da abordagem colaborativa de ensino e aprendizado porque todos os estudantes podem contribuir, enquanto os estudantes mais fracos podem se beneficiar de estarem expostos ao processo de pensamento dos alunos mais experientes. Exemplos de avaliação podem ser vistos a seguir:

- Quais são as vantagens de uma linguagem de programação em relação à outra?
- Avalie a linguagem C++ em relação à legibilidade.

4.2.1.8 Tarefas de Projeto

Tarefas de projeto exigem que os estudantes combinem o seu conhecimento para apresentar uma solução original e criativa. Tarefas de projeto podem ser solicitadas a um nível de implementação (por exemplo, escreva um código), ou a um nível conceitual (por exemplo, desenvolva um diagrama de classes).

O nível de implementação de projeto requer habilidades dos níveis cognitivos subjacentes (declarativo, compreensão, depuração, predição, e tarefas de provimento de exemplo). O nível de projeto conceitual só requer o conhecimento declarativo e de compreensão. Porém, um bom projeto conceitual requer compreensão pragmática desenvolvida sobre depuração, predição, criação de exemplo e tarefas de avaliação – os estudantes podem criar modelos de sistemas sem entender como implementá-los (por exemplo, o desenvolvimento de diagramas e nomes de métodos), mas, sem uma avaliação do que é exigido para implementar o sistema, a sua capacidade para construir projetos especialistas fica restringida. São exemplos de tarefas de projeto:

- Escreva um programa que usa um laço “for” para imprimir todos os números pares entre 100 e 2 em ordem inversa, por exemplo, 100, 98, 96, etc;
- Desenvolva um diagrama de casos de uso para um sistema de vídeo-locadora.

4.2.1.9 Tarefas de Resolução de Problemas

Tarefas de resolução de problemas poderiam ser consideradas outro modo de realizar tarefas de projeto, porém, porque os problemas impõem que o estudante responda a exigências de tarefas menos estruturadas, eles foram classificados como uma categoria

separada. Alguns estudantes podem talvez projetar sistemas com especificações conhecidas, mas são menos capazes de resolver problemas em que a abordagem não é bem definida. Tarefas de resolução de problemas promovem o desenvolvimento de contextualização de escopo, o pensamento crítico e a flexibilidade cognitiva que estão na área de comportamentos mais especialistas. São exemplos:

- Qual é o menor número que pode ser fatorado em 50 valores diferentes?
- Construa um método de ordenação mais eficaz que o método *QuickSort*.

4.2.1.10 Tarefas de reflexão

Tarefas de reflexão exigem que os estudantes avaliem os modos nos quais eles aprendem (ao invés de avaliar conteúdos). Isto ajuda no desenvolvimento de habilidades de controle, na capacidade para monitorar-se e avaliar decisões feitas durante o processo de resolução de problemas. Não obstante, reflexão também pode ajudar o aprendiz a se comprometer com os outros (quando a tarefa é colaborativa) em uma tentativa de melhorar a sua habilidade de aprender “com” e ensinar “para” outras pessoas. São exemplos:

- Reflita sobre as etapas do processo que você usou para completar esta tarefa;
- Reflita no modo em que você se ocupou com seu parceiro durante o processo de desenvolvimento de software em duplas.

4.2.2 Concluindo Sobre a Taxionomia de Tarefas de Bower

A hierarquia apresentada por Bower não descreve quais são as tarefas de nível mais alto que deveriam ser realizadas em momentos mais avançados do curso de programação e quais tarefas de níveis inferiores só deveriam acontecer no princípio. É importante que sejam encorajados que os estudantes iniciantes executem tarefas de ordem de pensamento mais alto,

embora em um peso menor e focalizando em material menos complexo que em fases posteriores da educação.

Não é proposto também que todas as tarefas estarão relacionadas nitidamente a uma única categoria da taxionomia; frequentemente, podem ser solicitados tipos de tarefas combinadas (por exemplo, “descreva um problema de performance no método de ordenação *ShellSort* e componha um exemplo original deste problema juntamente com uma possível solução”).

Destaca-se que a taxionomia de tarefas proposta por Bower torna o pedagogo consciente da gama de tarefas das quais os estudantes deveriam se ocupar para alcançar domínio na área. O modo no qual os níveis são hierarquicamente colocados, até certo ponto, acentua a importância do desenvolvimento cognitivo incremental.

4.3 A REVISADA TAXIONOMIA DE BLOOM APLICADA PARA MELHORAR O PROCESSO DE AVALIAÇÃO DOS ALUNOS EM PROGRAMAÇÃO

Thompson e colaboradores (2008) aplicaram a taxionomia de Bloom, revisada por Anderson e colaboradores (2001), com o intuito de melhorar o processo de avaliação dos alunos de programação. Segundo os autores, é difícil aplicar a taxionomia de Bloom a tarefas de avaliação em cursos introdutórios de programação, embora seja uma valiosa ferramenta com potencial para fomentar análise e discussão sobre avaliação em programação – se puder ser utilizada.

4.3.1 As Categorias da Revisada Taxionomia de Bloom e Exemplos de Tarefas Avaliativas em Programação

Foram discutidas todas as categorias da revisada taxionomia de Bloom com o objetivo de produzir uma interpretação consistente em conjunto com exemplos concretos que permitem aos pedagogos de informática utilizar a taxionomia para avaliar seus aprendizes de programação. Uma explicação de como as categorias podem ser aplicadas durante o processo de avaliação pode ser vista a seguir.

4.3.1.1 Lembrar

Segundo os autores, o ato de “lembrar” está definido como: “recuperação de conhecimento proveniente da memória de longo prazo”. Na taxionomia, esta categoria inclui reconhecimento e recordação. Tipos de atividades relacionadas ao Conhecimento, no contexto de programação, são os que se seguem:

- a) identificação de uma construção particular em um fragmento de código;
- b) reconhecimento da implementação de um conceito de uma área de estudo;
- c) reconhecimento da descrição apropriada para um conceito de uma área de estudo ou termo;
- d) recordação de qualquer conteúdo que faça parte do programa pedagógico. Este conteúdo poderia ser conhecimento efetivo. São exemplos: uma solicitação de uma definição conceitual; uma solicitação de um processo; uma solicitação de um algoritmo; uma solicitação de um padrão de projeto; uma solicitação de um algoritmo particular ou padrão de projeto implementado como uma solução de um problema específico no mesmo contexto dos exercícios propostos em sala de aula.

São Exemplos de atividades neste nível:

- Liste os operadores aritméticos em ordem crescente de precedência;
- Defina o propósito de um método construtor;
- Descreva o funcionamento do laço “while”.

Determinar se uma tarefa pertence a esta categoria requer conhecimento detalhado dos conteúdos do curso. Se a tarefa simplesmente puder ser completada através do processo de lembrança, a atividade pertence a esta categoria; caso contrário, tem que pertencer a uma das cinco categorias seguintes.

4.3.1.2 Entender

“Entender” está definido como: “construção de significado de mensagens instrutivas, inclusive mensagens orais, escritas e gráficas”. Na taxionomia, esta categoria inclui Interpretação, Exemplificação, Classificação, Sumarização, Dedução, Comparação e Explicação. Tipos de atividades relacionadas à compreensão no domínio de programação são os que se seguem:

- a) tradução de um algoritmo de uma forma de representação para outra forma (por exemplo, do português para a linguagem C);
- b) explicação de um conceito ou algoritmo ou padrão de projeto;
- c) apresentação de um exemplo de um conceito, algoritmo ou padrão de projeto.

Exemplo um:

- Explique o que o código abaixo faz:

```
void main() {  
    int i=0, j=1;  
    int vetor[10];  
    while(i<10) {  
        cin>>vetor[i];  
        i++;  
    }  
}
```

Exemplo dois:

- Os estudantes recebem um código fonte de uma classe e lhes é solicitado que: (1) Identifiquem o construtor definido nesta classe, transcrevendo a assinatura do construtor; e (2) Escrevam uma instrução que instancie (crie) um objeto que usa o construtor identificado.

Este exemplo se relaciona a duas categorias de processo cognitivo distintas. Primeiramente, os estudantes devem ser capazes de recordar as regras de sintaxe de uma definição de classe. Esta tarefa está relacionada ao nível de conhecimento. Em seguida, os estudantes devem ser capazes de deduzir a forma correta de realizar uma chamada a um método da classe baseada na assinatura da variável objeto instanciada.

4.3.1.3 Aplicar

“Aplicar” está definido como: “uso de um procedimento em uma determinada situação”. Na taxionomia, esta categoria inclui Execução e Implementação. Tipos de atividades relacionadas à aplicação no domínio de programação são os que se seguem:

- a) uso de processos e algoritmos ou padrões de projeto que o estudante conhece (podendo aplicá-los a um problema que está familiarizado e que não foi previamente resolvido no mesmo contexto, com os mesmos dados ou com as mesmas ferramentas);
- b) uso de processos e algoritmos ou padrões de projeto conhecidos pelo estudante e que podem ser aplicados a um problema pouco conhecido.

Exemplo um:

- Execute a expressão: $2 + 4 / 7 * 5 \% 3 == 7$

Este exemplo exige que o estudante siga um processo conhecido de aplicação das regras de precedência para avaliar a expressão. Se a expressão fosse extremamente simples, como “ $1 + 2$ ”, então seria esperado que o estudante a avaliasse usando apenas conhecimento. A complexidade da expressão exige que os estudantes sigam um algoritmo para computar os resultados. O processo exige que os estudantes compreendam as regras que ditam a ordem de precedência e avaliem a expressão executando as operações na ordem correta. A parte crítica da pergunta, em termos de aplicação, está relacionada com a categorização, em que os estudantes estão aplicando um processo para resolver o problema.

Exemplo dois:

- Aos estudantes, foi entregue o código fonte para uma classe de um Círculo. O código é semelhante a um exemplo usado no livro texto, mas, algumas características foram alteradas ou removidas no intuito de reduzir a quantidade de código. Foi também passado aos estudantes a classe Quadrado e a classe Triângulo. Cada classe tem a mesma estrutura de código, embora alguns atributos e métodos sejam distintos a cada uma delas. É solicitado que os estudantes: (1) Criem uma classe “Forma Geométrica” como uma superclasse destas três classes que contenha todos os atributos e métodos comuns; e (2) Reescrevam as classes Círculo, Quadrado e Triângulo de maneira que se tornem herdeiras da superclasse.

Este exemplo pertence ao nível Aplicação porque os estudantes foram apresentados ao processo de re-fatoração. É esperado que eles apliquem o processo de re-fatoração para desenvolver (criar) uma nova classe e, em seguida, outras três classes revisadas (reescrever). O uso dos verbos criar e reescrever neste contexto não insinua o senso de ser criativo como no nível Criação, pois não está sendo esperado que os estudantes desenvolvam um processo ou um algoritmo novo.

4.3.1.4 Analisar

“Analisar” está definido como: “separação de um objeto de estudo em suas partes constituintes e determinação de como estas partes se relacionam e qual o seu propósito na estrutura”. Na taxionomia, esta categoria inclui Diferenciação, Organização e Atribuição. Tipos de atividades relacionadas à análise no domínio de programação são os que se seguem:

- a) separação de uma tarefa de programação em diversos módulos (classes, componentes, funções, procedimentos, etc.);
- b) organização de módulos para alcançar um objetivo global;
- c) identificação de componentes críticos de um projeto de desenvolvimento;
- d) identificação de componentes sem importância ou exigências.

Exemplo:

- Dado o código para uma classe *Círculo*, é questionado aos estudantes: (1) o que é o método *Círculo* nesta classe? e (2) Como ele difere de outros métodos na classe?

É esperado que os estudantes provejam as seguintes respostas para o exemplo anterior: (1) “é um construtor” e (2) “é invocado automaticamente quando se cria uma instância de um objeto do tipo *Circulo*”. Diferentemente do exemplo dois do nível compreensão, em que é esperado apenas que os estudantes identifiquem o método na classe, neste exemplo é esperado que eles analisem um fragmento de código e expliquem o seu comportamento. A primeira parte da pergunta (o que é) envolve recordação de que um método com o mesmo nome de uma classe é uma construtora, levando o aluno a concluir que o método naquele caso é então um construtor. Na segunda metade da pergunta (como faz) é solicitado aos estudantes que diferenciem entre um construtor e outros métodos da classe. Diferenciar é uma das subcategorias do processo cognitivo de análise.

4.3.1.5 Avaliar

“Avaliar” está definido como: “fazer julgamentos baseados em critérios e padrões”. Na taxionomia, esta categoria inclui Verificação e Crítica. Tipos de atividades relacionadas à avaliação no domínio de programação são os que se seguem:

- a) determinar se um fragmento de código satisfaz às exigências através de uma estratégia de teste apropriada;
- b) criticar a qualidade de um fragmento de código baseado em padrões de codificação ou critérios de desempenho.

Exemplo:

- Os estudantes recebem dois fragmentos de código que possuem um laço de repetição e são usados para mostrar os números pares entre 0 e 100, como os códigos 1 e 2 a seguir.

Código 1:

```
for(int i=0; i<=100; i++) {  
    if (i % 2 == 0)  
        cout<<"O número "<< i <<" é par";  
}
```

Código 2:

```
for(int i=0; i<=100; i=i+2)  
{  
    cout<<"O número "<< i <<" é par";  
}
```

É então solicitado que os alunos avaliem qual deles é o código mais eficiente em termos de tempo de processamento. Não obstante, é solicitado que os alunos indiquem o motivo pelo qual aquele fragmento de código foi por eles identificado como mais eficiente.

A simples comparação da diferença sintática entre dois laços pertence ao nível compreensão. Quando é solicitado que os estudantes discutam qual método é mais eficiente, o

propósito da questão passa para o nível avaliação. Indicar o motivo pelo qual um fragmento de código foi classificado como mais eficiente que outro também é um processo cognitivo de avaliação.

4.3.1.6 Criar

“Criar” está definido como: “junção de elementos para formar um todo coerente ou funcional; reorganização de elementos em um novo padrão ou nova estrutura”. Na taxionomia, esta categoria inclui Geração, Planejamento, e Produção. Tipos de atividades relacionadas à criação no domínio de programação são os que se seguem:

- a) propor um algoritmo alternativo novo ou uma hipótese sobre uma nova combinação de algoritmos para resolver um problema;
- b) inventar um processo alternativo ou estratégia para resolver um problema; ou tarefas de programação complexas. Um bom exemplo poderia incluir a divisão de uma tarefa em sub-tarefas coesas menores para as quais seja possível aplicar algoritmos e processos conhecidos;
- c) construção de um segmento de código ou programa de um algoritmo inventado; ou aplicação de algoritmos conhecidos em uma combinação que é nova aos estudantes.

Exemplo:

- Crie uma classe String em C++ para suportar operações como concatenação, comparação, entre outras, a partir do uso de sobrecarga de operadores. Como exemplos, a classe criada pelo programador deve permitir que duas variáveis do tipo String possam ser comparadas com o uso do operador relacional “=” e uma concatenação entre variáveis do tipo String deve ser permitida usando o operador aritmético “+”.

A dificuldade com tarefas deste tipo é determinar se elas pertencem ao nível Aplicação ou Criação. O tamanho do problema influencia a sua dificuldade, mas não determina se é Aplicação ou Criação. A categoria criação deveria requerer pensamento criativo e a formação de um “todo coerente” ou funcional. Se os estudantes estão familiarizados com o algoritmo e a carga de processo cognitivo é inferior, então a pergunta deveria ser categorizada como Aplicação. É importante destacar que, para realizar atividades no nível de criação, os estudantes devem estar familiarizados com o processo de projeto de algoritmos.

A categoria cognitiva Criação se aplica nos casos em que o estudante não tem nenhuma familiaridade com o todo funcional. Portanto, eles não podem ter visto o algoritmo antes, mas podem ter visto fragmentos do algoritmo sendo utilizados para outras finalidades. Por outro lado, esta categoria requer conhecimento de um algoritmo e/ou processo e sua aplicação em uma determinada situação.

Em um programa de grande porte, por exemplo, podem existir tarefas que estão relacionadas à Aplicação (por exemplo, a aplicação de um padrão de projeto), enquanto o todo poderia estar relacionado à Criação – supondo que no projeto exista a necessidade de usar estratégias modernas e código inédito.

4.3.2 Concluindo Sobre a Taxionomia de Bloom Aplicada para Melhorar o Processo de Avaliação dos Alunos em Programação

O trabalho de Thompson e colaboradores (2008) descreveu efetivamente como utilizar a taxionomia de Bloom no intuito de melhorar o processo de avaliação de alunos em disciplinas de programação. Contudo, e embora não tenha sido esse o objetivo, os autores não se preocuparam em definir uma metodologia ou conjunto de linhas guia que proporcionem o ensino dos conhecimentos por eles exigidos para produzir bons programadores.

4.4 OUTROS TRABALHOS NA ÁREA

Doran e Langan (1995) relataram um projeto que implementou uma abordagem baseada no domínio cognitivo (usando a taxionomia de Bloom) nos dois primeiros anos de um curso de computação. Segundo os autores, foi utilizado sequenciamento de conteúdo em espiral de acordo com os níveis da taxionomia. O projeto também investigou o uso de trabalhos realizados em laboratórios, com avaliação frequente e uso de equipes desde o início. Eles usaram micro-objetivos de curso que foram classificados de acordo com níveis específicos na taxionomia de Bloom.

Howard, Carver e Lane (1996) se propuseram a identificar metas para várias tarefas (lições) de programação e classificá-las de acordo com os determinados níveis da taxionomia de Bloom. Segundo os autores, a maioria das lições tem várias metas de conhecimento relacionadas aos níveis da taxionomia que são alcançadas durante todo o curso de uma disciplina. Essas metas foram usadas para avaliar alunos de uma turma de programação em um curso de ciência da computação do Departamento de Engenharia Elétrica e Computação da Academia Militar Norte Americana (EUA).

Thompson, Hunt e Kinshuk (2006) desenvolveram objetivos de aprendizagem de estruturas de dados baseados na taxionomia de objetivos educacionais revisada por Anderson e colaboradores (2001). O trabalho dos autores deu origem às pesquisas de Thompson e colaboradores (2008).

Acreditando que o uso de taxionomias genéricas não é eficiente para ensino de programação e disciplinas de computação, Fuller e colaboradores (2007) desenvolveram uma taxionomia de aprendizado específica para conteúdos em Ciência da Computação. Em seus estudos, os autores separaram objetivos específicos e os classificaram em: objetivos relacionados à atividade de interpretação; e objetivos relacionados à atividade de produção (construção/criação).

Ademir de Jesus e Raabe (2009) discutiram como cada uma das categorias da taxionomia vêm sendo interpretadas e utilizadas em avaliações nas disciplinas de “Programação de Computadores” e aplicaram as idéias discutidas na elaboração de um

instrumento de avaliação (com questões exemplo) que possa ser utilizado por outros professores e pesquisadores.

4.5 CONCLUINDO SOBRE OS TRABALHOS RELACIONADOS ENCONTRADOS

Conforme dito anteriormente, os trabalhos encontrados na literatura se ocuparam de aplicar taxionomias genéricas (ou próprias) no ensino de programação de computadores e de conteúdos em computação. Embora os trabalhos descritos nesta seção foram selecionados como exemplos de referencial teórico para o presente trabalho, nenhuma menção foi feita nesses estudos (e em nenhum outro estudo que encontramos) em relação à existência de um método de ensino de programação baseado em alguma dessas taxionomias. Não obstante, nenhum dos trabalhos encontrados se preocupou em analisar ou aplicar a taxionomia de Bloom no domínio afetivo para o ensino de conteúdos em computação, seja em “Programação de Computadores”, seja em qualquer outra disciplina.

CAPÍTULO 5

MÉTODO TRIFÁSICO DE ENSINO- APRENDIZAGEM (MTEA)

Método é um conjunto dos meios dispostos convenientemente para alcançar um fim e especialmente para chegar a um conhecimento científico ou comunicá-lo aos outros. Pode ser considerado também como: (1) a ordem ou sistema que se segue no estudo ou no ensino de qualquer disciplina; (2) a maneira sistemática de fazer as coisas; (3) o modo de proceder; (4) a maneira de dizer, de fazer, de ensinar algo, segundo certos princípios e em determinada ordem (KÖCHE, 2000).

O MTEA se encaixa nos conceitos anteriormente citados e, portanto, deve ser considerado um método de ensino-aprendizagem baseado na taxionomia de objetivos educacionais de Bloom (segundo o domínio cognitivo e afetivo). Optou-se pela produção e descrição de tal método em concordância com Bower (2008), ao questionar o fato de que os educadores em informática têm investido muita energia definindo conteúdos programáticos, culminando em pouca ênfase no processo de ensino (o que explica a percebida carência de métodos para ensino de programação), e ao defender a importância do processo de ensino-aprendizagem no intuito de disseminar o conhecimento.

5.1 POR QUE USAR O DOMÍNIO COGNITIVO ?

A complexidade dos conteúdos nas disciplinas de programação em cursos de computação é elevada. Isto ocorre principalmente por exigirem alto índice de abstração por

parte dos aprendizes. Não obstante, as diversas áreas de atuação na computação, principalmente as relacionadas ao desenvolvimento de software, estão se expandindo a um ritmo acelerado, tornando cada vez mais difícil o domínio individual de uma ou várias delas. Por estas e outras razões, formular objetivos comportamentais precisos, passíveis de observação e mensuração, para aprendizes de programação de computadores, assume importância crescente. A elaboração de tais objetivos:

- a) Delimita a tarefa e retira toda a ambiguidade e dificuldade de interpretação dos enunciados;
- b) Assegura a possibilidade de medição, de modo que a qualidade e efetividade da experiência de aprendizado pode ser determinada;
- c) Permite que o professor e aprendizes possam distinguir entre as diferentes variedades ou classes de comportamentos, possibilitando então que eles decidam qual estratégia de aprendizado tem maiores chances de ser ótima;
- d) Fornece um sumário completo e sucinto do curso, que pode servir como estrutura conceitual ou como “organizadores avançados” para o aprendizado.

Não obstante, conforme visto anteriormente, o domínio cognitivo está relacionado aos conteúdos que se pretende expor ao aprendiz e à maneira pela qual o mesmo pode absorvê-los, desde a simples aquisição do conhecimento até a habilidade de criação de novos fatos ou artefatos a partir de conhecimentos prévios. Como é interesse deste trabalho desenvolver um método de ensino que, espera-se, seja eficaz, considerou-se necessário mapear todo o conteúdo programático da disciplina “Programação de Computadores” e definir como as categorias de conhecimento, segundo o domínio cognitivo, serão alcançadas em cada conteúdo mapeado.

Por fim, existe uma grande preocupação neste trabalho em estimular situações de aprendizagem (previamente planejadas) baseadas em conflitos cognitivos específicos, diretamente relacionadas às habilidades que se espera de bons programadores.

A busca de objetivos cognitivos fez com que o planejamento deixasse de ser visto como uma organização burocrática e sim como um instrumento que estimula de um lado, a competência de ensinar: relacionar conteúdos a objetivos cognitivos, e de outro, situações de aprendizagem. Dessa forma, conclui-se que é dever dos professores e de projetistas de ensino criar ambientes de aprendizagem que estimulem conflitos cognitivos significativos nos alunos e, para tanto, os professores

devem conhecer os conteúdos que lecionam e quais os objetivos cognitivos que os conteúdos promovem (MENDES, 2002, p. 83).

5.2 POR QUE USAR O DOMÍNIO AFETIVO ?

Muitos esforços têm sido relatados na literatura com o objetivo de promover melhorias no ensino-aprendizagem de “Programação de Computadores”. Entretanto, não foram encontrados estudos que abordaram a questão afetiva dos aprendizes neste contexto. Este achado contraria as expectativas de Bloom e colaboradores (1956) quando defendem não haver nenhuma separação fundamental entre o domínio cognitivo e afetivo.

Scheerer (1954, apud BLOOM, 1964) vai ainda mais longe. Segundo o autor:

O comportamento pode ser conceituado como sendo encastrado numa matriz motivacional-emocional-cognitiva, na qual não é possível uma separação verdadeira. Não importa como seja dividido o comportamento, os ingredientes de cognição-emoção-motivação estão presentes, numa ou noutra ordem.

Seguindo essa linha de raciocínio, sugere-se a importância de considerar os aspectos afetivos relacionados aos cognitivos quando se pretende elaborar um método que privilegie o ensino-aprendizagem de qualquer conteúdo.

5.3 DOMÍNIO COGNITIVO VERSUS AFETIVO

Pressupõe-se então um estreito laço entre os domínios cognitivo e afetivo, enquanto é defendida a teoria de que o domínio afetivo pode afetar o domínio cognitivo (positivamente ou negativamente) e vice-versa.

A pesquisa resumida por Jacob (1957, apud BLOOM, 1964) levanta questões a respeito da defensabilidade desta pressuposição. O autor sustenta o ponto de vista de que, sob

algumas condições, o desenvolvimento de comportamentos cognitivos pode construir ou destruir certos comportamentos afetivos desejados. Por outro lado, alguns comportamentos afetivos podem facilitar ou dificultar o desenvolvimento de comportamentos cognitivos.

Corroborando com Jacob, salienta-se que uma abordagem errônea para atingir os objetivos cognitivos em um determinado conteúdo ou disciplina pode afetar, direta e negativamente, a aceitação e satisfação dos alunos em relação àquele conteúdo, culminando em uma crescente desmotivação. Não obstante, uma abordagem errônea para atingir os objetivos afetivos em relação ao curso de uma disciplina, tornará a tarefa do estudo enfadonha e não motivadora, culminando em bloqueio e dificuldade em adquirir conhecimento naquela área.

5.4 MÉTODO TRIFÁSICO DE ENSINO-APRENDIZAGEM

O Método Trifásico de Ensino-Aprendizagem foi desenvolvido pensando-se em um sistema trifásico de aquisição de conhecimento elaborado especificamente para este trabalho (Figura 5.1). Analisando o trabalho de Bloom sob o domínio cognitivo, agrupou-se o aprendizado em três fases que os aprendizes devem transpassar para adquirir conhecimento “útil” sobre um determinado assunto. A fase inicial é identificada pela aquisição de conhecimento – quando o aluno se dá conta das informações que conceituam algo e compreende seu funcionamento. A fase intermediária identifica a etapa em que o aprendiz se dá conta dos motivos que explicam o conhecimento e dos motivos pelos quais é importante adquiri-lo – quando o aluno se torna capaz de fazer uma reflexão crítica sobre aquele conhecimento e aplicá-lo nas situações cabíveis. A fase final é identificada pela habilidade de usar aquele conhecimento para um objetivo maior, que pode culminar na criação de algo ou na geração de novos conhecimentos. Nesta fase, o aprendiz também é capaz de avaliar se o seu conhecimento ou sua criação é diferencial em algum aspecto se comparado aos conhecimentos ou criações similares. As fases que compõem esse sistema foram denominadas respectivamente como se segue: (1) Fase 1 – Vejo, Escuto e Compreendo; (2) Fase 2 – Discuto, Aplico e Reflito; e (3) Fase 3 – Pratico, Faço e Crio.

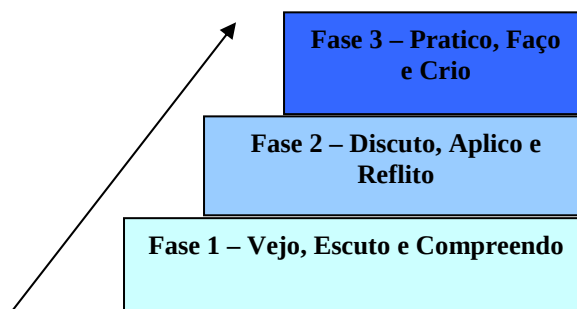


Figura 5.1 – Fases do Método Trifásico de Ensino-Aprendizagem.

Durante a primeira fase, devem ser elaboradas as atividades a fim de atingir os objetivos educacionais para as categorias Conhecimento e Compreensão do domínio cognitivo em conjunto com os objetivos educacionais para a categoria Acolhimento do domínio Afetivo de acordo com todo o conteúdo programático da disciplina. Destaca-se que as atividades descritas para os objetivos afetivos deverão estar voltadas para permitir ou privilegiar que sejam atingidos os objetivos cognitivos daquele conteúdo.

Durante a segunda fase, são elaboradas as atividades a fim de atingir os objetivos educacionais para as categorias Aplicação e Análise do domínio cognitivo em conjunto com os objetivos educacionais para as categorias Resposta e Valorização do domínio Afetivo de acordo com todo o conteúdo programático da disciplina.

Durante a terceira fase, são elaboradas as atividades a fim de atingir os objetivos educacionais para as categorias Síntese e Avaliação do domínio cognitivo em conjunto com os objetivos educacionais para as categorias Organização e Caracterização por um Valor ou Complexo de Valores do domínio Afetivo de acordo com todo o conteúdo programático da disciplina.

Embora todos os objetivos afetivos possam ocorrer durante a realização de cada uma das fases descritas anteriormente, ao relacionar os objetivos cognitivos com os afetivos, percebe-se que o Acolhimento ocorre predominantemente durante a primeira fase, enquanto a Resposta e Valorização ocorrem predominantemente durante a segunda fase e a Organização e Caracterização por um Valor ou Complexo de Valores ocorrem com maior frequência na terceira fase.

Depois de elaboradas as atividades propostas para cada uma das três fases de aprendizagem de uma determinada disciplina, o professor deve segui-las ao explicar o conteúdo programático em sua sala de aula.

É importante destacar que o MTEA pode ser utilizado em qualquer disciplina alvo. Como o presente estudo tem seu foco na disciplina “Programação de Computadores”, foram desenvolvidas tabelas de aplicação do método que apresentam todas as atividades (especificamente de acordo com o conteúdo programático dessa disciplina) a serem realizadas pelo professor e pelos aprendizes durante cada uma das três fases (capítulo 6).

Embora o conhecimento de uma unidade indivisível de conteúdo siga o modelo linear proposto por Bloom e colaboradores (1956), o conhecimento amplo desse conteúdo segue um modelo espiral (como na Figura 5.2), em que é possível transpassar uma mesma fase do MTEA diversas vezes para adquirir conhecimento em temas distintos (inter-relacionados ou não) dentro dele (do conteúdo).

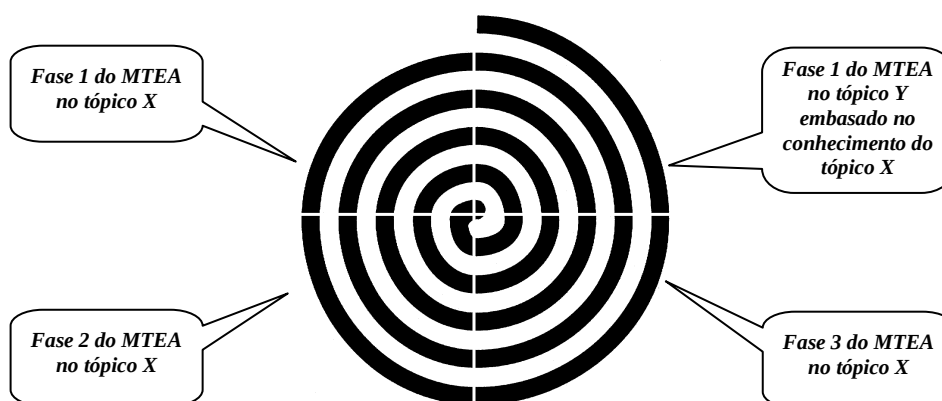


Figura 5.2 – Adquirindo Conhecimento de Conteúdos em Forma de Espiral.

5.5 CHECKLIST PARA TESTAR O MTEA

Para avaliar a efetividade do MTEA, foi produzida uma *checklist* baseada nos processos e verbos de ação associados às fases do Método Trifásico de Ensino-Aprendizagem (Tabelas 5.1, 5.2 e 5.3). O processo de checagem é simples e é composto pelas seguintes atividades:

1. Toda tarefa proposta aos alunos deve ser produzida de acordo com o nível ao qual a mesma pertence. Assim, uma tarefa irá medir se os alunos estão aptos a transpassar apenas a primeira, segunda ou terceira fase do MTEA e, portanto, está relacionada a uma única tabela de *checklist* (Tabela 5.1, 5.2 ou 5.3);
2. O professor deve relacionar cada tarefa proposta aos alunos com um ou mais verbos de ação de acordo com a sua tabela de *checklist* correspondente. Se a tarefa se enquadra na fase 1, é necessário relacioná-la a um ou mais verbos de ação vistos na tabela 5.1. Se a tarefa se enquadra na fase 2, é necessário relacioná-la a um ou mais verbos de ação vistos na tabela 5.2. E, por fim, se a tarefa se enquadra na fase 3, é necessário relacioná-la a um ou mais verbos de ação vistos na tabela 5.3;
3. Pode ser criada uma *checklist* para cada aluno, solicitando-o preenchimento. Contudo, como parâmetro para decisão sobre o momento ideal de troca de fase de aprendizado, é indicado que o professor produza uma única *checklist* para a classe como um todo;
4. À medida que a turma resolver uma tarefa proposta, o professor deverá marcar a opção “☐ Ok” de cada verbo de ação previamente relacionado àquela tarefa;
5. É de livre escolha do professor decidir quando a turma está apta a transpassar uma fase de aprendizado, embora seja aconselhado que isto ocorra somente quando todos os verbos de ação de uma tabela de *checklist* forem selecionados, demonstrando que aquela turma se capacitou a resolver problemas relacionados àqueles verbos de ação e, consequentemente, àquela fase de aprendizado;
6. É importante destacar que uma *checklist* corresponde ao conjunto das três Tabelas (5.1, 5.2 e 5.3) de *checklist* vistas a seguir.

Tabela 5.1 – Checklist para Testar o MTEA na Fase 1 (de Acordo com Processos e Verbos de Ação)

Fase	Processos ou Verbos de Ação Associados
Fase 1 – Vejo, Escuto e Compreendo	Definir ☐ Ok
	Reconhecer ☐ Ok
	Recitar ☐ Ok
	Identificar ☐ Ok
	Rotular ☐ Ok
	Compreender ☐ Ok
	Examinar ☐ Ok
	Mostrar ☐ Ok
	Coletar ☐ Ok
	Listar ☐ Ok
	Traduzir ☐ Ok
	Interpretar ☐ Ok
	Explicar ☐ Ok
	Descrever ☐ Ok
	Resumir ☐ Ok
	Demonstrar ☐ Ok

Tabela 5.2 – Checklist para Testar o MTEA na Fase 2 (de Acordo com Processos e Verbos de Ação)

Fase	Processos ou Verbos de Ação Associados
Fase 2 – Discuto, Aplico e Reflito	Aplicar ☐ Ok
	Solucionar ☐ Ok
	Experimentar ☐ Ok
	Demonstrar ☐ Ok
	Construir (sem a intenção de ser criativo) ☐ Ok
	Mostrar ☐ Ok
	Fazer ☐ Ok
	Ilustrar ☐ Ok
	Registrar ☐ Ok
	Conectar ☐ Ok
	Relacionar ☐ Ok
	Diferenciar ☐ Ok
	Classificar ☐ Ok
	Arranjar ou Estruturar ☐ Ok
	Agrupar ☐ Ok
	Interpretar ☐ Ok
	Organizar ☐ Ok
	Categorizar ☐ Ok
	Retirar ☐ Ok
	Comparar ☐ Ok
	Dissecar ☐ Ok
	Investigar ☐ Ok

Tabela 5.3 – Checklist para Testar o MTEA na Fase 3 (de acordo com Processos e Verbos de Ação)

Fase	Processos ou Verbos de Ação Associados
Fase 3 – Prático, Faço e Crio	Projetar ☐ Ok
	Reprojetar ☐ Ok
	Combinar ☐ Ok
	Consolidar ☐ Ok
	Agregar ☐ Ok
	Compor ☐ Ok
	Formular Hipótese ☐ Ok
	Construir ☐ Ok
	Traduzir (no sentido criativo) ☐ Ok
	Imaginar ☐ Ok
	Inventar ☐ Ok
	Criar ☐ Ok
	Inferir ☐ Ok
	Produzir ☐ Ok
	Predizer ☐ Ok
	Interpretar ☐ Ok
	Verificar ☐ Ok
	Julgar ☐ Ok
	Criticar ☐ Ok
	Decidir ☐ Ok
	Discutir ☐ Ok
	Verificar ☐ Ok
	Disputar ☐ Ok
	Escolher ☐ Ok

5.6 OBSERVAÇÕES IMPORTANTES

Embora a taxionomia de Bloom descreve coerentemente seis etapas de aprendizado, conforme visto anteriormente, procurou-se agrupar as etapas que de maneira intrínseca se relacionam fortemente – formando um Método Trifásico de Ensino-Aprendizagem. Considerou-se que “Conhecimento” e “Compreensão” estão tão fortemente entrelaçados que é difícil separá-los durante a explanação de um conteúdo programático de programação – embora possam ser mais facilmente distinguidos em outros contextos. Não obstante, percebeu-se que os aprendizes de programação dificilmente executam a atividade de “Aplicação” ou “Análise” sem a presença concomitante da outra – é muito comum a necessidade de conhecimento em nível de análise para que o aprendiz consiga aplicar um

conhecimento de programação (quando esse processo de análise falha ou não ocorre, o que se vê são programas que terminam por não solucionar os problemas ou gerar outros maiores). Em contra partida, análises são, na maioria das vezes, realizadas no intuito de ajudar o aprendiz a aplicar seus conhecimentos. Por fim, é esperado que programadores hábeis sejam capazes de avaliar o próprio produto de software baseados na maioria dos critérios conhecidos sobre qualidade de software. Em contra partida, a habilidade de avaliar (conforme requisitos de eficácia e eficiência) se faz primordial para a produção de bons softwares. É importante destacar também maior facilidade e melhor didática em se trabalhar com um conjunto menor de etapas de aprendizado, porém mais fortemente relacionadas.

Ao serem submetidos ao MTEA, os aprendizes devem estar bem informados de cada etapa de desenvolvimento durante a explanação dos tópicos do conteúdo programático. Isso coloca um pouco de disciplina ao processo e os torna mais conscientes das fases do próprio aprendizado.

É proposto que as etapas sejam transpassadas em unidades de aulas distintas com o objetivo de gerar oportunidade aos aprendizes de solucionar exercícios práticos ou pesquisas conceituais extraclasse. Assim, é fortemente aconselhado nunca transpassar mais de uma etapa para um conteúdo programático em uma única aula.

CAPÍTULO 6

APLICANDO O MTEA NA DISCIPLINA “PROGRAMAÇÃO DE COMPUTADORES”

Como exemplo, o seguinte capítulo descreve as atividades modeladas para aplicação do MTEA na disciplina “Programação de Computadores”. É interesse deste trabalho atingir os objetivos do domínio cognitivo através dos objetivos afetivos. Como complemento, é importante ressaltar que os objetivos afetivos são impulsionados, neste exemplo, pelo uso da técnica de Programação em duplas (Figura 6.1).

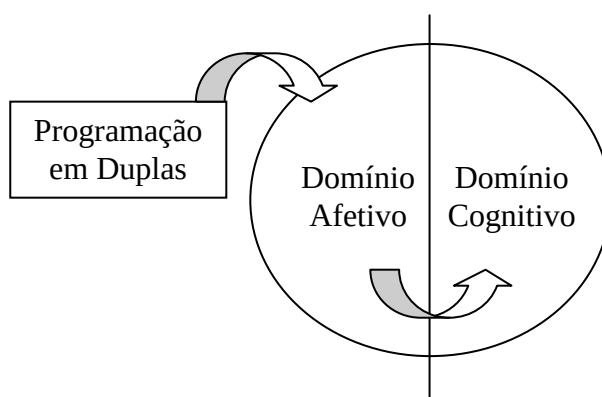


Figura 6.1 – Usando a Programação em Duplas Para atingir os Objetivos Educacionais nos Domínios Afetivo e Cognitivo.

A Programação em Duplas estimula conflitos sócio-cognitivos. Conflitos, se bem empregados, podem ser uma boa ferramenta para promover reflexão e articulação – agindo, assim, como gatilhos de mudanças cognitivas (TEDESCO, 2001, abstract). Contudo, é esperado que o uso de Programação em Duplas promova não somente mudanças cognitivas (devido ao conflito sócio-cognitivo), mas também mudanças afetivas positivas, ocasionadas por um maior senso de realização e aprendizado. Essas duas crenças, inclusive, podem salientar o diferencial entre o presente trabalho e os demais encontrados na literatura – a teoria

de que a motivação influencia em maior possibilidade de aprendizado e maior aprendizado torna os aprendizes motivados a buscar mais conhecimento.

6.1 O MTEA NA DISCIPLINA “PROGRAMAÇÃO DE COMPUTADORES”

O conteúdo programático da disciplina “Programação de Computadores” pode ser visto na tabela 6.1. É importante destacar que este é um exemplo de conteúdo programático utilizado em uma Universidade específica cujos parâmetros curriculares foram baseados na Sociedade Brasileira de Computação, onde foram realizados os experimentos com o MTEA. Portanto, é esperado que outras instituições acadêmicas possuam variações (não muito discrepantes) de conteúdos programáticos para suas disciplinas consideradas correspondentes.

Tabela 6.1 – Conteúdo Programático da Disciplina “Programação de Computadores”

Objetivos Específicos		Temas
Desenvolvimento Sob a Técnica de Aprendizado Individual	Compreender os sistemas de numeração e as transformações de base	Introdução aos Sistemas de Numeração
		Transformação de Decimal para Binário
		Transformação de Decimal para Octal
		Transformação de Decimal para Hexadecimal
		Transformação de uma Base b qualquer para Decimal
	Introduzir Algoritmos	Introdução aos Algoritmos e Noções de Lógica
		Algoritmizando a Lógica
		Noções de Compiladores
		Elaboração de Comentários
	Construir expressões e entender a prioridade de operadores	Expressões
		Prioridade de operadores
		Operadores aritméticos, relacionais e lógicos
	Definir dados	Definição de dados
		Tipos primitivos
		Identificadores
		Palavras-chave
		Constantes
		Variáveis
		Comando de atribuição (transferência de valores)
		Comandos de entrada e saída
	Construir tipos abstratos de dados	Conversões entre tipos
		Tipos homogêneos (vetores e matrizes)
		Tipos heterogêneos (vetores e matrizes)

Tabela 6.1 – Conteúdo Programático da Disciplina “Programação de Computadores”

<<Continuação>>

Objetivos Específicos		Temas
Desenvolvimento Sob a Técnica de Aprendizado em Duplas	Conhecer os tipos de estruturas de controle e entender as estruturas de sequência	Tipos de estruturas de controle
		Estruturas de sequência
		Blocos de instruções (escopo)
	Entender as estruturas de seleção (Alternativas)	Seleção simples
		Seleção composta
		Seleção encadeada heterogênea
		Seleção encadeada homogênea
		Seleção de múltipla escolha
	Entender as estruturas de repetição (Laços)	Repetição com teste no início (repetir enquanto)
		Repetição com teste no final (repetir até que)
		Repetição com teste programado (repetir para)
	Entender modularização e construir software modular	Função
		Procedimento
		Passagem de parâmetros
		Recursividade

6.1.1 Compreender os Sistemas de Numeração e as Transformações de Base

O conteúdo programático “Sistemas de Numeração e Transformações de Base” forma a base sobre a qual os sistemas de computação foram construídos. O computador nada mais é do que uma máquina digital destinada ao processamento de números. Ainda que o insumo do processamento seja representado por um conjunto de caracteres alfabéticos, um conjunto de pontos de uma imagem ou qualquer outro formato, ele sempre ocorrerá em dados formatados numericamente (zeros e uns – os bits). Esta representação não é casual. O valor “0” representa o valor negativo da corrente elétrica, enquanto o valor “1” representa o valor positivo.

Por se tratar de um conteúdo essencialmente prático, é sugerido o uso do recurso “quadro branco” durante sua explanação – É interessante que o professor descreva os sistemas de numeração e demonstre as transformações de base “dinamicamente” durante a aula (se possível, buscando a participação de alunos que se interessem em ajudar nas transformações). Este tipo de estratégia traz o aprendiz para dentro do processo de ensino-aprendizagem, ajudando-o a atingir os estágios iniciais (Acolhimento, Resposta e Valorização) do domínio

afetivo. – Transparências podem ser utilizadas, mas aconselha-se sua utilização apenas como referencial à teoria do conteúdo em questão.

Este é o primeiro contato que o aprendiz tem com o conteúdo da disciplina “Programação de Computadores”. Por este motivo, é interessante que o professor se dedique com maior afinco no sentido de tornar a aula bastante agradável. Levantar questionamentos sobre conceitos e teorias, solicitar a participação dos aprendizes sempre que um tema novo for abordado e incentivá-los a buscar informações em outras fontes de pesquisa como, por exemplo, a Internet, têm um enorme potencial para ajudá-los a atingir os objetivos segundo os domínios cognitivo e afetivo.

A tabela 6.2 mostra o que pode ser feito para que o aluno transpasse a fase inicial (“Vejo, Escuto e Compreendo”) do corrente conteúdo programático.

Tabela 6.2 – Objetivos e Habilidades da Fase “Vejo, Escuto e Compreendo” para o Objetivo Específico “Compreender os Sistemas de Numeração e as Transformações de Base”

Como Encaminhar o Aluno ao Conhecimento e Compreensão?	Como Levar o Aluno ao Acolhimento?	Habilidade(s) a Ser(em) Adquirida(s)
- Descrever o significado e importância dos sistemas de numeração. - Apresentar um breve histórico sobre o sistema decimal de numeração. - Descrever como se transforma um número em decimal para uma base binária e vice-versa. - Descrever como se transforma um número em decimal para uma base octal e vice-versa. - Descrever como se transforma um número em decimal para uma base hexadecimal e vice-versa. - Descrever a importância de se usar bases divisíveis por 2 ao estudar sistemas computacionais.	- Iniciar a aula a partir do questionamento: “O que é um sistema de numeração?”. - Levantar outros questionamentos como, por exemplo: (1) “Por que nosso sistema de numeração é decimal?”; (2) “Qual é o sistema de numeração dos computadores e por que não é decimal?”. - Solicitar que os alunos realizem buscas na Internet por curiosidades sobre cada um dos sistemas de numeração abordados em sala de aula e apresentem (pelo menos uma curiosidade por aluno) no início da próxima aula.	- Capacidade de descrever informações conceituais sobre os sistemas de numeração como, por exemplo: “Por que utilizar um sistema de numeração?”. - Capacidade de reconhecer em que sistema de numeração um número está representado. - Capacidade de reconhecer a importância de um sistema de numeração em contextos (ciência ou área) específicos.

O aluno nesta fase deve ser capacitado a recuperar conhecimento proveniente da memória de longo prazo e a conhecer o que está sendo comunicado, fazendo uso dele, sem necessariamente relacioná-lo a outro material ou perceber suas implicações mais completas.

O conhecimento sobre sistemas de numeração é importante porque o aprendiz está adentrando em uma ciência cuja lógica matemática é baseada em apenas dois valores (sim e não / verdadeiro e falso / zeros e uns) e todas as formas de armazenamento ou tratamento de informação (nesta ciência) devem estar relacionadas a uma potência de “2” (como, por exemplo, os sistemas de numeração octal e hexadecimal), diferentemente do sistema decimal usado no cotidiano.

A habilidade de aplicar esse conhecimento é adquirida na medida em que o aprendiz transpassa a fase intermediária (“Discuto, Aplico e Reflito”) do corrente conteúdo programático (Tabela 6.3).

Tabela 6.3 – Objetivos e Habilidades da Fase “Discuto, Aplico e Reflito” para o Objetivo Específico “Compreender os Sistemas de Numeração e as Transformações de Base”

Como Encaminhar o Aluno à Aplicação e Análise?	Como Levar o Aluno à Resposta e Valorização?	Habilidade(s) a Ser(em) Adquirida(s)
- Demonstrar diversas transformações de base (Decimal → Binário / Octal / Hexadecimal → Decimal). - Descrever como se transforma um número decimal para uma base b qualquer. - Descrever como se transforma um número em uma base b qualquer para um número decimal. - Demonstrar o relacionamento de cada algarismo (por exemplo, o algarismo 6 do número 625 na base hexadecimal corresponde a $6 \cdot 16^2$) com o seu valor na base decimal. - Comparar a quantidade de algarismos de um número qualquer na base decimal com o mesmo número representado nas demais bases. - Discutir o motivo pelo qual cada uma das bases abordadas em sala é importante no seu contexto.	- Solicitar que os alunos façam uma análise do motivo pelo qual bases menores geram números com maior quantidade de algarismos. - Solicitar que os alunos discutam e descrevam o motivo pelo qual os números hexadecimais são acrescidos dos algarismos A, B, C, D, E e F. - Solicitar que os alunos discutam e expliquem questões como, por exemplo: “O motivo pelo qual 1 byte corresponde a 8 bits ao invés de 10”; “O motivo pelo qual 1 Kbyte contém 1024 bytes ao invés de 1000”. - Solicitar que os alunos respondam a questões do tipo: “Quantos números podem ser representados por 1 byte? Justifique sua resposta.”; “Descreva os 16 primeiros números (em base binária) que 1 byte pode representar”.	- Capacidade de transformar um número decimal em qualquer outra base não demonstrada em sala de aula (e vice-versa). - Capacidade de reconhecer e explicar as diferenças estruturais de um número representado em cada uma das bases abordadas em sala. - Capacidade de descrever quantos números podem ser representados por uma quantidade específica de algarismos (em qualquer sistema de numeração).

Nesta fase, o aluno deve ser capaz de fazer uso de um procedimento em uma determinada situação (técnica de transformação de base) e de separar um objeto de estudo em

suas partes constituintes a fim de determinar como estas partes se relacionam e qual o seu propósito na estrutura.

Em uma fase final de aprendizado, a fase “Pratico, Faço e Crio”, é esperado que os alunos reconheçam situações em que o domínio exige um sistema de numeração diferente (ainda não estudado), identifiquem esse sistema requerido e as regras que definem suas operações aritméticas e gerem conhecimento novo baseado nele. A tabela 6.4 descreve como direcionar o aluno a alcançar essa fase final de conhecimento para o corrente conteúdo programático.

Tabela 6.4 – Objetivos e Habilidades da Fase “Pratico, Faço e Crio” para o Objetivo Específico “Compreender os Sistemas de Numeração e as Transformações de Base”

Como Encaminhar o Aluno à Síntese e Avaliação?	Como Levar o Aluno à Organização e Caracterização por um Valor ou Complexo de Valores?	Habilidade(s) a Ser(em) Adquirida(s)
- Demonstrar como seria um sistema de numeração que privilegiasse única e exclusivamente a contagem dos minutos e segundos (Sistema Sexagesimal). - Demonstrar as operações aritméticas (adição, subtração, multiplicação e divisão) nas diversas bases.	- Elaborar situações adversas em que seja necessário um sistema de numeração específico e solicitar que os alunos descrevam a base deste sistema e justifiquem sua resposta (para cada situação elaborada). - Disponibilizar uma lista de exercícios que contenha valores representados em diversas bases e solicitar que os alunos descrevam o seu valor correspondente em outra base. - Solicitar que os alunos descrevam como resolver operações aritméticas entre números de bases diferentes. - Solicitar que os alunos respondam (e justifiquem) questões do tipo: “Qual é mais rápido? – Um computador que trabalha na base binária ou um que trabalha na base octal?”. - Solicitar que os alunos descrevam um sistema de computador que utilize um sistema de numeração ternário (positivo, negativo e outro valor imaginário qualquer) e demonstrem as operações aritméticas neste sistema.	- Capacidade de reconhecer que sistema de numeração é mais indicado em uma situação específica. - Capacidade de realizar operações aritméticas nas diversas bases e entre valores representados em bases diferentes. - Capacidade de criar sistemas de numeração para situações específicas juntamente com suas regras de operações aritméticas.

Deste momento em diante o aluno se torna apto a fazer julgamentos baseados em critérios e padrões e a juntar elementos, reorganizando-os em um novo padrão ou nova estrutura que corresponda a uma criação própria.

Exemplos de exercícios para este e todos os conteúdos programáticos (classificados de acordo com as etapas do Método Trifásico de Ensino-Aprendizagem) podem ser vistos no Apêndice A.

6.1.2 Introduzir Algoritmos

O conteúdo programático “Introduzir Algoritmos” corresponde ao primeiro contato que realmente proporciona aos aprendizes a sensação de que estão sendo acometidos a uma nova maneira de enxergar o funcionamento sistemático das coisas e fatos. Esta fase costuma ser determinante para que o aluno construa um sentimento de adoração ou de aversão à disciplina “Programação de Computadores”.

Embora a educação nos níveis primário e médio abordem a lógica em diversas disciplinas, não é exigido constantemente que os alunos sejam capazes de algoritmizá-la, mas apenas em raras situações como, por exemplo, durante o curso da disciplina “matemática”, quando é solicitado que os alunos resolvam problemas com sistemas de equações. Analisando o exemplo seguinte, torna-se fácil perceber como a solução de um sistema de equações segue um determinado tipo de algoritmização.

- A soma das idades de André e Carlos é 22 anos. Descubra as idades de cada um deles, sabendo-se que André é 4 anos mais novo do que Carlos.

- Solução: Primeiro traduz-se o problema para a linguagem matemática. A letra “c” será usada para representar a idade de Carlos, enquanto a letra “a” deve representar a idade de André, logo:

$$a = c - 4 \text{ e } c + a = 22$$

- Assim, aplicando o sistema de equações “ $a = c - 4$ ” e “ $c + a = 22$ ”, tem-se:

$$c + (c - 4) = 22$$

$$2c - 4 = 22$$

$$2c = 22 + 4$$

$$2c = 26$$

$$c = 13$$

- Como:

$$a = c - 4$$

$$a = (13) - 4$$

$$a = 9$$

- Resposta: Carlos tem 13 anos e André tem 9 anos.

O problema anterior foi solucionado executando o seguinte algoritmo:

- Ler e interpretar o enunciado
- Construir as equações que a interpretação do enunciado permitir: **$c + a = 22$** e **$a = c - 4$**
- Aplicar a segunda equação na primeira: **$c + (c - 4) = 22$**
- Obter o resultado da primeira equação: **$c = 13$**
- Aplicar o resultado da primeira equação na segunda: **$a = (13) - 4$**
- Obter o resultado da segunda equação: **$a = 9$**

Ao ler o algoritmo proposto anteriormente, tem-se a sensação de que construir algoritmos é algo trivial. Contudo, é importante considerar alguns conhecimentos e habilidades que tornam essa tarefa difícil para alunos iniciantes. É preciso que os aprendizes, especificamente neste caso:

- Saibam interpretar corretamente o problema proposto;
- Tenham conhecimento sobre como montar equações;
- Saibam onde pesquisar problemas correlatos;
- Saibam reconhecer elementos encontrados em problemas correlatos que possam ajudá-los a solucionar o novo problema;
- Tenham habilidade em sistematizar problemas – ou separá-los em pequenos módulos para resolvê-los individualmente a fim de gerar uma solução final;
- Tenham enorme interesse em descobrir e construir soluções – ao contrário daqueles que procuram encontrar soluções já prontas;
- Sejam persistentes sob qualquer adversidade.

É importante destacar que a lista de conhecimentos e habilidades necessárias à construção de algoritmos pode crescer exponencialmente de acordo com a complexidade do problema a solucionar – o que realmente ocorre na maioria dos casos. Em observações anteriores, durante anos de docência desta disciplina, ficou notória a predisposição dos aprendizes em desistir de problemas quando eles os identificam como complexos ao invés de insistir na construção de uma solução própria.

Finalizando, o trabalho com este conteúdo programático deve privilegiar que os aprendizes adquiram gosto (satisfação ou prazer) por solucionar problemas. A partir deste momento, o aluno deve ser capaz de enxergar o mundo de um ponto de vista sistematizado e deve se sentir bem diante desta habilidade de analisar minuciosamente o funcionamento das coisas e fatos.

A tabela 6.5 descreve como direcionar o aluno a transpassar a fase “Vejo, Escuto e Compreendo” do corrente conteúdo programático.

Tabela 6.5 – Objetivos e Habilidades da Fase “Vejo, Escuto e Compreendo” para o Objetivo Específico “Introduzir Algoritmos”

Como Encaminhar o Aluno ao Conhecimento e Compreensão?	Como Levar o Aluno ao Acolhimento?	Habilidade(s) a Ser(em) Adquirida(s)
- Descrever o conceito de algoritmo. - Descrever o conceito de lógica matemática. - Demonstrar como algoritmizar a lógica matemática. - Desenvolver algoritmos propostos em sala de aula, descrevendo o raciocínio elaborado para se obter cada componente necessário como, por exemplo, os ingredientes, os talheres, as louças e panelas necessárias para produzir um bolo. - Descrever as diferenças entre algoritmos matemáticos (passíveis de implementação computacional) e algoritmos cotidianos (como o exemplo da produção de um bolo). - Descrever os conceitos de compiladores e apresentar aos alunos códigos fonte de programas (algoritmos em uma linguagem compreendida pelo computador)	- Iniciar a aula solicitando que os alunos realizem uma busca na Internet sobre o conceito de Algoritmo (para isso, escolha iniciar este conteúdo durante uma aula de laboratório) – Lance o seguinte desafio: “a melhor resposta, segundo avaliação do professor, será premiada em 1 ponto extra”. Todos os alunos deverão ler suas respostas em voz alta, sendo em seguida comentadas pelo professor. - Solicitar que os alunos produzam algoritmos do tipo: “Algoritmo para fritar um ovo”; “Algoritmo para limpar um carro”; “Algoritmo para estudar para uma prova”. - Solicitar que os alunos respondam o seguinte questionamento: “Por que os algoritmos dos colegas não são iguais ao seu, mas podem também ser considerados corretos?”.	- Capacidade de perceber que todas as ações (humanas ou não) são realizadas através de pequenas atividades dispostas sequencialmente. - Capacidade de compreender que as regras de sequenciamento para produção de um bolo, por exemplo, se aplicam a qualquer algoritmo computacional. - Capacidade de compreender a sistematização necessária para realização de uma tarefa qualquer. - Capacidade de produzir algoritmos detalhados para a solução de qualquer problema do cotidiano, sem a necessidade de que sejam tomadas decisões de correção de percurso.

sendo executados por compiladores. O compilador pode ser um simples interpretador de linguagem algorítmica (Portugol) ou uma linguagem de programação como Pascal, C++, etc.		
--	--	--

Alguns (se não muitos) estudantes, ao se depararem com este “novo” paradigma de solução sistematizada de problemas, acabam por criar uma imagem negativa da disciplina de “Programação de Computadores” devido à criteriosidade (ou minuciosidade) exigida para se construir bons algoritmos. Contribuir para que os aprendizes desenvolvam o prazer em produzir soluções algorítmicas é um desafio a ser vencido pelos professores a partir deste exato instante. Pode ser interessante solicitar a participação dos alunos na solução dos algoritmos (em forma de requisição oral). É importante valorizar a maneira como o aluno raciocina sobre determinados aspectos a fim de encaminhá-lo na produção de uma solução aceita para aquele problema. Ou seja, é fundamental que o professor dê crédito às soluções propostas pelos alunos.

Na próxima fase (“Discuto, Aplico e Reflito”), é esperado que os alunos se tornem capazes de aplicar a técnica de solução sistematizada em problemas do seu cotidiano. A tabela 6.6 descreve como direcionar o aluno a transpassá-la de acordo com o corrente conteúdo programático.

Tabela 6.6 – Objetivos e Habilidades da Fase “Discuto, Aplico e Reflito” para o Objetivo Específico “Introduzir Algoritmos”

Como Encaminhar o Aluno à Aplicação e Análise?	Como Levar o Aluno à Resposta e Valorização?	Habilidade(s) a Ser(em) Adquirida(s)
- Disponibilizar algoritmos prontos (como solução de enunciados pré-definidos) aos aprendizes e realizar, no quadro branco, uma análise do motivo pelo qual cada instrução ou conjunto de instruções foi ou foram utilizada(s) naquela solução. - De posse de algoritmos prontos, alterar trechos de código e descrever uma análise sobre o efeito causado como consequência de cada alteração. - Descrever alguns efeitos colaterais que podem ocorrer em algoritmos quando são submetidos a alterações	- Solicitar que os alunos façam uma análise nos algoritmos de outros colegas. Os alunos devem ser incentivados a completar ou alterar o que for (por eles) considerado necessário no algoritmo analisado. - Distribuir enunciados com algoritmos semiprontos aos alunos e solicitar-lhes: (1) uma análise do motivo pelo qual foram utilizadas algumas estruturas (pré-definidas) dentro do algoritmo; (2) a conclusão do algoritmo. - Distribuir algoritmos prontos e solicitar que os alunos: (1) façam	- Capacidade de analisar algoritmos e sugerir alterações ou incrementos sem a explícita intenção de torná-los mais eficientes. - Capacidade de separar um algoritmo em diversos fragmentos que serão interpretados e considerados (futuramente) como módulos. - Capacidade de organizar fragmentos de algoritmos para alcançar um objetivo global. - Capacidade de identificar componentes críticos de um

estruturais. Faça uma descrição dos elementos que são acometidos pelos efeitos colaterais e uma análise do motivo pelo qual ocorrem. - Descrever como produzir bons comentários. Dar exemplos de bons e péssimos comentários. Descrever a importância dos comentários como valor agregado na qualidade do algoritmo. - Em qualquer demonstração de algoritmo no quadro branco, descrever os elementos críticos àquela solução e analisar seu impacto local e global. Descrever também aqueles elementos não considerados críticos ou que podem ser facilmente substituídos sem impacto significativo no resultado final.	uma análise para compreender o propósito de cada algoritmo; (2) elaborem um enunciado para cada algoritmo. - Solicitar que todo aluno elabore enunciados de algoritmos para uma ciência qualquer (por exemplo, a física, a química, a matemática, etc.). Estes enunciados devem ser distribuídos pelo professor aos demais alunos como tarefa avaliativa. A nota atribuída a essa tarefa pode ser considerada pontuação extra. - Distribuir enunciados e algoritmos prontos e solicitar que os alunos descrevam que elementos (neles) encontrados podem ser utilizados para produzir um novo algoritmo que represente a solução de um enunciado pré-definido (diferente dos demais, porém, correlacionado).	algoritmo. - Capacidade de identificar componentes sem importância significativa em um algoritmo. - Capacidade de usar processos ou padrões de projeto que o estudante conhece para produzir novos algoritmos. - Capacidade de construir bons comentários relativos às instruções (ou conjunto de instruções utilizadas no algoritmo) ou ao contexto do enunciado. - Capacidade de analisar algoritmos incompletos e produzir fragmentos de código que os leve à completude.
--	---	--

Até o presente momento, os alunos devem estar capacitados a construir algoritmos do cotidiano em um nível de complexidade baixo. Contudo, na medida em que os problemas se tornam mais complexos, pode ser exigido que os programadores adquiram habilidades necessárias à construção de algoritmos mais eficientes, otimizados e, alguns casos, inovadores. Essas habilidades são adquiridas quando os aprendizes transpassam a fase final “Prático, Faço e Crio” do corrente conteúdo programático (Tabela 6.7).

Tabela 6.7 – Objetivos e Habilidades da Fase “Prático, Faço e Crio” para o Objetivo Específico “Introduzir Algoritmos”

Como Encaminhar o Aluno à Síntese e Avaliação?	Como Levar o Aluno à Organização e Caracterização por um Valor ou Complexo de Valores?	Habilidade(s) a Ser(em) Adquirida(s)
- Disponibilizar algoritmos prontos incorretos e descrever os motivos pelos quais foram assim considerados. Realizar (nos mesmos algoritmos) as correções necessárias. Comparar as duas soluções lado a lado no quadro branco. - Disponibilizar algoritmos diferentes para a solução de um mesmo enunciado e compará-los em termos de eficiência (seja em relação ao tamanho – em linhas de código –,	- Disponibilizar algoritmos diferentes para a solução de um único enunciado e solicitar que os alunos descrevam que algoritmo é mais eficiente e justifiquem sua resposta. - Disponibilizar algoritmos para a solução de diversos problemas e solicitar que os alunos produzam um algoritmo mais eficiente para cada um deles (em quantidade de linhas de código e em consumo de recursos	- Capacidade de criar soluções algorítmicas inovadoras para problemas do cotidiano. - Capacidade de avaliar soluções algorítmicas propostas por terceiros. - Capacidade de reconhecer os diversos critérios que podem ser utilizados para medir a eficiência de um algoritmo. - Capacidade de reconhecer os

seja em relação ao consumo de recursos de processamento). - Descrever como encontrar problemas cotidianos que se encaixam como ótimos candidatos a serem solucionados através do uso de algoritmos. - Descrever as variáveis a serem analisadas (e como analisá-las) quando se deseja elaborar soluções inovadoras ou mais eficientes para problemas já resolvidos.	de processamento). - Solicitar que os alunos construam soluções algorítmicas para problemas ainda não abordados em sala de aula. - Solicitar que os alunos avaliem (em termos de eficiência) soluções algorítmicas (para um determinado problema) dos colegas de sala. - Solicitar que os alunos avaliem os comentários agregados a um algoritmo disponibilizado pelo professor. - Solicitar que os alunos avaliem os comentários agregados a um algoritmo desenvolvido por um colega de turma (pré-definido).	diversos critérios que podem ser utilizados para medir a qualidade semântica dos comentários agregados ao algoritmo. - Capacidade de reconhecer construções algorítmicas incorretas (sintática ou semanticamente) e sugerir as correções necessárias. - Capacidade de reconhecer construções algorítmicas ineficientes e propor melhorias.
---	--	--

Exemplos de exercícios para este conteúdo programático podem ser encontrados no Apêndice A.2.

6.1.3 Construir Expressões e Entender a Prioridade de Operadores

A maioria dos programas de computador executa cálculos aritméticos (DEITEL e DEITEL, 2001). Os cálculos aritméticos são representados por expressões aritméticas como, por exemplo, $A = B * 5$. Essas expressões são construídas valendo-se do conjunto de caracteres interpretáveis pela máquina e são formadas por operandos e operadores (multiplicação, divisão, adição, subtração, mod, div e potência). Porém, os programas não são formados apenas por expressões aritméticas, mas por uma mistura pré-definida pelos programadores de expressões lógicas, relacionais, literais e aritméticas. Expressões lógicas são aquelas formadas por operadores lógicos (não, e, ou) e operandos do tipo lógico (verdadeiro ou falso). Expressões relacionais são aquelas formadas por operadores relacionais (maior, menor, maior igual, menor igual, igual e diferente) e operandos do tipo numérico (embora possam existir operações relacionais sobre caracteres como, por exemplo, a comparação de igualdade entre nomes de indivíduos). Expressões literais, por sua vez, são aquelas que executam alguma alteração em um conjunto de caracteres como, por exemplo, a concatenação entre o nome e o sobrenome de um indivíduo outrora armazenados, cada um

deles, em locais diferentes (locais estes que serão definidos na próxima seção como variáveis).

As expressões podem ser formadas por mais de um operando (de tipos equivalentes ou não) e podem ser mescladas para construir operações mais complexas como, por exemplo: $(5 * 6) > 17$. Por este motivo, existe uma ordem de prioridade de operadores que deve ser respeitada pelos computadores quando lhes é ordenada a execução dessas expressões. Compreender os conceitos das expressões aritméticas, lógicas, relacionais, literais e da ordem de prioridade de operadores compõe a fase inicial (“Vejo, Escuto e Compreendo”) de aprendizagem do conteúdo “Construir Expressões e Entender a Prioridade de Operadores” e é fundamental para a construção de programas cujo processamento pode ser considerado confiável – embora este não seja o único fator que define a confiabilidade de um programa.

A tabela 6.8 mostra o que pode ser feito para que o aluno transpasse essa fase em relação ao presente conteúdo programático.

Tabela 6.8 – Objetivos e Habilidades da Fase “Vejo, Escuto e Compreendo” para o Objetivo Específico “Construir Expressões e Entender a Prioridade de Operadores”

Como Encaminhar o Aluno ao Conhecimento e Compreensão?	Como Levar o Aluno ao Acolhimento?	Habilidade(s) a Ser(em) Adquirida(s)
- Descrever o que são expressões aritméticas, lógicas, relacionais e literais. - Descrever a ordem de prioridade de operadores. - Descrever o conceito do símbolo parênteses nas expressões e o modo como ele é utilizado para alterar a ordem de prioridade dos operadores. - Construir expressões aritméticas no quadro branco e executá-las (passo a passo) de acordo com a ordem de prioridade dos operadores aritméticos. - Construir expressões lógicas e relacionais no quadro branco e executá-las (passo a passo) de acordo com a ordem de prioridade dos operadores lógicos e relacionais. - Construir expressões literais no quadro branco e executá-las (passo a passo).	- Solicitar que os alunos resolvam listas de exercícios com expressões aritméticas de acordo com a ordem de prioridade de operadores. - Solicitar que os alunos resolvam listas de exercícios com expressões lógicas de acordo com a ordem de prioridade de operadores. - Solicitar que os alunos resolvam listas de exercícios com expressões relacionais de acordo com a ordem de prioridade de operadores. - Solicitar que os alunos resolvam listas de exercícios com expressões literais de acordo com a ordem de prioridade de operadores. - Solicitar que os alunos resolvam listas de exercícios com expressões mistas de acordo com a ordem de prioridade de operadores.	- Capacidade de identificar o tipo de uma expressão apresentada pelo professor. - Capacidade de resolver expressões aritméticas, lógicas, relacionais, literais e mistas de acordo com a ordem de prioridade de operadores.

- Construir expressões mescladas no quadro branco e executá-las passo a passo de acordo com a ordem de prioridade entre tipos de operadores e entre operadores.		
---	--	--

Em alguns casos, as expressões aritméticas podem ser apresentadas como parte dos enunciados de problemas propostos, em outros, pode ser necessário que os programadores sejam capazes de compô-las através da interpretação desses enunciados. Não obstante, poderão haver casos em que as expressões serão definidas através do tipo de problema proposto como, por exemplo, quando ao programador for solicitado o desenvolvimento de um algoritmo para calcular as raízes de uma equação do 2º grau (cujas expressões necessárias são: (1) $\Delta = b^2 - 4 * a * c$; (2) $x' = (-b + \text{raiz}(\Delta)) / (2 * a)$; e (3) $x'' = (-b - \text{raiz}(\Delta)) / (2 * a)$). Por esses motivos, é importante que o aprendiz seja capaz de interpretar e analisar os diversos problemas no intuito de identificar ou construir as equações necessárias à sua solução. Essa habilidade é adquirida durante a fase “Discuto, Aplico e Reflito” do presente conteúdo programático. A tabela 6.9 mostra o que pode ser feito para que o aluno transpasse essa fase.

Tabela 6.9 – Objetivos e Habilidades da Fase “Discuto, Aplico e Reflito” para o Objetivo Específico “Construir Expressões e Entender a Prioridade de Operadores”

Como Encaminhar o Aluno à Aplicação e Análise?	Como Levar o Aluno à Resposta e Valorização?	Habilidade(s) a Ser(em) Adquirida(s)
- Descrever enunciados de problemas matemáticos que podem ser resolvidos através do sistema de equações e solucioná-los no quadro branco. - Descrever problemas que exigem o uso de expressões específicas e solucioná-los no quadro branco. - Descrever problemas das diversas ciências, como por exemplo, o problema do cálculo do movimento retilíneo uniformemente variado (MRUV) da física, e resolvê-los no quadro através da identificação e representação das expressões necessárias definidas na literatura específica.	- Solicitar que os alunos interpretem enunciados de problemas (dispostos em uma lista de exercícios) e identifiquem as expressões necessárias à solução de cada problema. - Solicitar que os alunos solucionem problemas semiprontos, disponibilizados com expressões pré-definidas incompletas. - Solicitar que os alunos realizem análise para verificar a correteza de expressões pré-definidas para problemas (As expressões consideradas por eles como incorretas devem ser reescritas corretamente).	- Capacidade de interpretar enunciados de problemas e construir expressões necessárias à solução dos mesmos. - Capacidade de analisar o contexto de um problema e identificar as expressões necessárias à sua solução de acordo com a literatura disponível na área à qual o problema se aplica. - Capacidade de identificar incorreções em expressões pré-disponibilizadas.

Conhecer os conceitos sobre expressões e ser capaz de identificá-las nos enunciados de problemas propostos é importante para que o aprendiz seja capaz de construir expressões corretas para problemas novos (ou novas expressões para a solução de problemas já resolvidos). Não obstante, programadores devem ser capazes de avaliar e comparar expressões com o objetivo de identificar aquelas consideradas mais eficientes ou de otimizá-las. Essas capacidades são adquiridas quando o aluno transpassa a fase final (“Pratico, Faço e Crio”) do presente conteúdo programático (Tabela 6.10).

Tabela 6.10 – Objetivos e Habilidades da Fase “Pratico, Faço e Crio” para o Objetivo Específico “Construir Expressões e Entender a Prioridade de Operadores”

Como Encaminhar o Aluno à Síntese e Avaliação?	Como Levar o Aluno à Organização e Caracterização por um Valor ou Complexo de Valores?	Habilidade(s) a Ser(em) Adquirida(s)
- Descrever expressões diferentes que solucionam um mesmo problema e avaliar e comparar a eficiência destas expressões. - Descrever expressões que solucionam problemas e demonstrar como elas podem ser otimizadas (produzindo expressões menores e mais eficientes).	- Disponibilizar diferentes expressões para a solução de um problema e solicitar que os alunos identifiquem a expressão mais eficiente ou mais otimizada. - Solicitar que os alunos construam enunciados de problemas de acordo com o sistema de equações, bem como suas soluções.	- Capacidade de criar problemas de acordo com o sistema de equações. - Capacidade de otimizar expressões. - Capacidade de definir que expressão, dentre muitas, é mais eficiente para a solução de um determinado problema.

Exemplos de exercícios para este conteúdo programático podem ser encontrados no Apêndice A.3.

6.1.4 Definir dados

Conforme dito na seção 6.1.1, “...O computador nada mais é do que uma máquina digital destinada ao processamento de números...”. Estes números, em um nível de abstração mais elevado, são interpretados por programadores como “dados”. Os dados neste nível de abstração podem armazenar diferentes valores além dos valores primários “0” e “1”. Compreender os tipos de dados no nível de abstração dos programadores é essencial para a

produção de algoritmos especificamente desenvolvidos para uma dentre diversas finalidades. Esses dados são armazenados em elementos denominados variáveis ou constantes que são representados por identificadores (apelidos a serem usados pelos programadores para manipular os dados). Por este motivo, construir bons identificadores (semanticamente corretos – que tenham relação direta com a representação do dado) é primordial para ajudar na produção e manutenção de algoritmos.

O conhecimento dos conceitos deste conteúdo programático abre o caminho para que o aprendiz se torne capaz de resolver problemas passíveis de solução computacional – diferentes dos problemas cotidianos até então estudados como, por exemplo, o “problema do algoritmo para fritar ovos”.

É importante destacar que os dados primitivos podem variar de um compilador de linguagem de programação para outro (por exemplo, o compilador “Turbo C++” não reconhece o tipo “lógico” e, portanto, não é capaz de armazenar os valores “verdadeiro” e “falso”. Contudo, em outros compiladores, como o “Turbo Pascal”, este tipo primitivo se encontra já incluso). Assim, é importante destacar os tipos primitivos comuns, os tipos inclusos na linguagem utilizada nas aulas de laboratório e salientar que existem variações de uma linguagem à outra.

A tabela 6.11 descreve como direcionar o aluno a transpassar a fase “Vejo, Escuto e Compreendo” do conteúdo programático “Definir Dados”.

Tabela 6.11 – Objetivos e Habilidades da Fase “Vejo, Escuto e Compreendo” para o Objetivo Específico “Definir Dados”

Como Encaminhar o Aluno ao Conhecimento e Compreensão?	Como Levar o Aluno ao Acolhimento?	Habilidade(s) a Ser(em) Adquirida(s)
- Descrever os conceitos de dados (“O que são dados?”; “Por que utilizá-los?”; “Quais os tipos primitivos existentes e que valores se restringem a representar?”). - Descrever o que são identificadores e como produzi-los de maneira que possam ser considerados significativos. - Descrever o que são e quais são as palavras-chave em algoritmo ou em uma linguagem de programação	- No início da aula, solicitar que os alunos realizem uma busca na Internet sobre os conceitos de dados e tipos primitivos de dados (Esta aula deve ser ministrada em laboratório). - Solicitar que os alunos realizem uma busca na Internet para encontrar a lista de palavras-chave das linguagens de programação: Pascal e C++ (e/ou da linguagem usada na disciplina “Programação de	- Capacidade de compreender a importância dos dados na produção de algoritmos passíveis de implementação computacional. - Capacidade de identificar a necessidade do uso de comandos de entrada de dados, bem como os tipos de dados a serem manipulados neste processo. - Capacidade de identificar a

específica que não podem ser utilizados como identificadores. - Descrever o conceito de constantes e variáveis. - Descrever o conceito de manutenibilidade de algoritmos e estabelecer a ligação entre a manutenibilidade e o uso de constantes. - Exemplificar situações em que o uso de constantes privilegia a manutenibilidade de algoritmos. - Descrever o processo de interpretação de enunciados (ou problema proposto) no intuito de identificar as variáveis necessárias à produção de um algoritmo. - Descrever e exemplificar comando de atribuição e conversão entre tipos de dados. - Descrever e exemplificar comandos de entrada e saída de dados.	Computadores” na instituição em que a XLP estiver sendo implementada). - Solicitar que os alunos recorram a livros de programação e descrevam duas regras básicas, neles encontradas, para a construção de bons identificadores. - Solicitar que os alunos descrevam prováveis variáveis que se relacionam a domínios ou contextos específicos (por exemplo: “Descreva prováveis variáveis de um algoritmo para calcular a nota média final dos alunos da disciplina matemática”; “Descreva prováveis variáveis de um algoritmo para calcular as raízes de uma equação do segundo grau”).	necessidade de construir variáveis para solução de um problema específico. - Capacidade de identificar a necessidade do uso de constantes. - Capacidade de realizar conversão entre tipos.
---	---	--

O conhecimento da estrutura dos tipos de dados primitivos é fundamental para a construção de algoritmos que se aplicam, por exemplo, à área comercial – os conhecidos sistemas de informação. Em um nível mais elevado de conhecimento, esses dados podem ser, em geral, combinados para aplicação em situações específicas. Este conhecimento pode ser adquirido na fase “Discuto, Aplico e Reflito” do corrente conteúdo programático (tabela 6.12).

Tabela 6.12 – Objetivos e Habilidades da Fase “Discuto, Aplico e Reflito” para o Objetivo Específico “Definir Dados”

Como Encaminhar o Aluno à Aplicação e Análise?	Como Levar o Aluno à Resposta e Valorização?	Habilidade(s) a Ser(em) Adquirida(s)
- Descrever situações em que o dado a ser processado é restrito a um único tipo como, por exemplo: “Situações em que é necessário armazenar o salário de um indivíduo requerem que a variável para tal finalidade seja do tipo real (que possa armazenar um valor com casa decimal)”; “Situações em que é necessário verificar se um número é divisível por outro requerem que a variável para tal finalidade seja do	- Solicitar que os alunos analisem situações diversas como, por exemplo: “Quando é importante usar tipo de dado inteiro?; “Quando não é permitido usar tipo de dado real?”; “Quando é semanticamente incorreto usar tipo de dado inteiro?”. - Solicitar que os alunos discutam o motivo pelo qual palavras-chave não podem ser utilizadas como identificadores em algoritmos ou em	- Capacidade de identificar palavras-chave que não podem ser utilizadas como identificadores (variáveis ou constantes). - Capacidade de identificar situações em que o uso de constantes é indicado no intuito de melhorar a manutenibilidade do algoritmo.

tipo inteiro”; “Situações em que é necessário armazenar um valor de status (tipo verdadeiro ou falso) são mais indicadas a serem implementadas com o uso de variável do tipo lógico”; etc. - Descrever situações em que a conversão entre tipos é necessária como, por exemplo, quando o programador precisa converter um dado CPF (armazenado em forma de caracteres) para valor numérico no intuito de realizar cálculos de validação.	uma linguagem de programação específica. - Disponibilizar enunciados de algoritmos e solicitar que os alunos identifiquem e descrevam as variáveis necessárias à sua solução. - Disponibilizar enunciados de algoritmos e solicitar que os alunos identifiquem os atributos (valores) candidatos a serem representados como constantes. - Disponibilizar enunciados de problemas que exigem algum tipo de conversão de dados e cálculos de validação como, por exemplo: “Algoritmo que valida um número de cartão de crédito” – Solicitar que os alunos descrevam as variáveis e os passos necessários à sua solução.	- Capacidade de analisar semanticamente a qualidade de um identificador no contexto em que está inserido. - Capacidade de identificar os casos nos quais a conversão entre tipos é necessária.
--	---	--

Em um terceiro nível de conhecimento, na fase “Pratico, Faço e Crio”, é exigido que os aprendizes sejam capazes de criar algoritmos inovadores para a solução de problemas pré-especificados e bem definidos. É importante destacar que todos os algoritmos solicitados aos alunos neste instante podem ser considerados inovadores, uma vez que ainda não realizaram e não lhes foram demonstrados algoritmos semelhantes. Ainda neste terceiro nível, os aprendizes precisam adquirir as habilidades necessárias para avaliar a qualidade semântica dos identificadores criados nos algoritmos. A tabela 6.13 mostra como levar o aluno a transpassá-lo.

Tabela 6.13 – Objetivos e Habilidades da Fase “Pratico, Faço e Crio” para o Objetivo Específico “Definir Dados”

Como Encaminhar o Aluno à Síntese e Avaliação?	Como Levar o Aluno à Organização e Caracterização por um Valor ou Complexo de Valores?	Habilidade(s) a Ser(em) Adquirida(s)
- Construir diversos identificadores para representar uma única variável e compará-los semanticamente de acordo com critérios específicos pré-definidos. - Construir algoritmos mais complexos em que seja necessário o uso de variáveis de todos os tipos primitivos. - Construir algoritmos compostos por pequenos algoritmos já descritos	- Distribuir enunciados de problemas com uma lista de identificadores necessários à sua solução e solicitar que os alunos relacionem todos os identificadores com o(s) respectivo(s) enunciado(s). - Distribuir enunciados de problemas com uma lista de identificadores necessários para a solução de cada problema e solicitar que os alunos avaliem semanticamente os identificadores em relação ao	- Capacidade de relacionar identificadores a problemas (de acordo com a semântica dos identificadores e o contexto dos problemas). - Capacidade de produzir identificadores significativos para o contexto em que serão inseridos. - Capacidade de construir algoritmos com comandos de

em sala de aula que formam uma solução de um problema mais elaborado (ou complexo).	contexto de cada enunciado. - Distribuir algoritmos prontos e solicitar que os alunos avaliem semanticamente a qualidade dos identificadores e reescrevam aqueles reprovados em sua avaliação. - Solicitar que os alunos elaborem um enunciado para um problema cujo algoritmo exija a implementação de exatas 5 variáveis.	atribuição e comandos de entrada e saída de dados.
---	---	--

Exemplos de exercícios para este conteúdo programático podem ser encontrados no Apêndice A.4.

6.1.5 Construir Tipos Abstratos de Dados

Os tipos de dados estudados no conteúdo programático da seção 6.1.4 representam os dados na sua forma mais primitiva. Esses dados primitivos são usados para abstrair atributos ou características simples de objetos, lugares, fatos, pessoas, etc. Quando o objetivo de um algoritmo envolve o processamento de dados compostos (como, por exemplo, o armazenamento de todas as informações pessoais de um cliente) é necessário que o programador construa estruturas capazes de representá-los. Essas estruturas são conhecidas como tipos abstratos de dados.

Tipos abstratos de dados podem ser compostos por valores de um mesmo tipo primitivo (como, por exemplo, vetores ou matrizes de inteiros), também conhecidos como tipos abstratos homogêneos, ou por valores de tipos diferentes (como, por exemplo, vetores ou matrizes de uma estrutura que armazena dados compostos como nome, endereço, telefone, etc.), também conhecidos como tipos abstratos heterogêneos.

O conhecimento sobre a construção dos tipos abstratos de dados homogêneos e heterogêneos é adquirido na fase inicial de aprendizado (“Vejo, Escuto e Compreendo”) do presente conteúdo programático e é primordial para a produção de algoritmos que processam informações comerciais (sistemas de informação, sistemas de apoio à decisão, entre outros) e

de algoritmos destinados a processamentos matemáticos (planilhas eletrônicas, calculadoras, entre outros).

A tabela 6.14 descreve o que fazer para que o aluno transpasse a fase inicial do conteúdo programático “Construir Tipos Abstratos de Dados”.

Tabela 6.14 – Objetivos e Habilidades da Fase “Vejo, Escuto e Compreendo” para o Objetivo Específico “Construir Tipos Abstratos de Dados”

Como Encaminhar o Aluno ao Conhecimento e Compreensão?	Como Levar o Aluno ao Acolhimento?	Habilidade(s) a Ser(em) Adquirida(s)
- Descrever os conceitos de abstração. - Descrever os conceitos de tipos abstratos de dados homogêneos (Vetores e Matrizes). Descrever os conceitos de tipos abstratos de dados heterogêneos (Vetores e Matrizes). - Demonstrar a transferência de valores para dados abstratos homogêneos (Vetores e Matrizes). - Demonstrar a transferência de valores para dados abstratos heterogêneos (Vetores e Matrizes). - Demonstrar a entrada e saída de valores em dados abstratos homogêneos (Vetores e Matrizes). - Demonstrar a entrada e saída de valores em dados abstratos heterogêneos (Vetores e Matrizes).	- Solicitar que os alunos construam tipos abstratos de dados homogêneos para representar vetores e matrizes de inteiros, reais e caracteres com tamanho pré-definido pelo professor. - Solicitar que os alunos construam tipos abstratos de dados heterogêneos para representar vetores e matrizes que armazenam, por exemplo, dados pessoais, dados de veículos, etc., com características e tamanho pré-definidos pelo professor. - Solicitar que os alunos produzam algoritmos para ler e mostrar informações armazenadas em variáveis dos tipos de dados construídos nos dois exercícios propostos anteriormente. - Solicitar que os alunos realizem cálculos aritméticos diversos nos dados numéricos das variáveis dos tipos de dados construídos nos exercícios propostos anteriormente.	- Capacidade de construir tipos abstratos de dados homogêneos a partir de uma modelagem de dados pré-definida em um enunciado de um problema. - Capacidade de construir tipos abstratos de dados heterogêneos a partir da modelagem de dados pré-definida em um enunciado de um problema. - Capacidade de realizar operações de entrada e saída, operações de atribuição e aritméticas em variáveis de tipos abstratos de dados homogêneos. - Capacidade de realizar operações de entrada e saída, operações de atribuição e aritméticas em variáveis de tipos abstratos de dados heterogêneos.

O conhecimento sobre a construção de tipos abstratos de dados deve ser considerado ainda mais importante quando o programador se torna capaz de: (1) identificar as situações em que existe tal necessidade; (2) modelar os tipos abstratos necessários ao contexto em questão. A tabela 6.15 descreve o que fazer para que o aluno transpasse a fase intermediária (“Discuto, Aplico e Reflito”) do presente conteúdo programático, em que o aprendiz se capacita a realizar essas duas atividades.

Tabela 6.15 – Objetivos e Habilidades da Fase “Discuto, Aplico e Reflito” para o Objetivo Específico “Construir Tipos Abstratos de Dados”

Como Encaminhar o Aluno à Aplicação e Análise?	Como Levar o Aluno à Resposta e Valorização?	Habilidade(s) a Ser(em) Adquirida(s)
- Demonstrar casos reais em que o uso de tipos abstratos de dados homogêneos é necessário (por exemplo: “Programa que lê os valores das idades de cada aluno da sala, calcule a idade média entre os alunos e mostre todos alunos cuja idade se apresenta abaixo da média”; “Programa que lê os valores das notas de matemática de cada aluno da sala, calcule a nota média e mostre todos alunos cuja nota se apresenta acima da média”). - Demonstrar casos reais em que o uso de tipos abstratos de dados heterogêneos é necessário (por exemplo: “Programa que lê os dados de um veículo e emite o CRV – Certificado de Registro do Veículo”; “Programa que lê os dados pessoais dos clientes e emite uma lista de clientes com seus respectivos endereços para produção de uma mala direta promocional de produtos”).	- Disponibilizar algoritmos produzidos por alunos para solução dos enunciados do exercício anterior e solicitar que os alunos analisem a corretude e a necessidade dos tipos criados e sugiram alterações (justificadas). - Disponibilizar enunciados de problemas em que sejam necessários tipos abstratos de dados homogêneos e solicitar que os alunos construam algoritmos para efetuar operações de entrada e saída nesses dados. O aluno deverá ser capaz de identificar, sozinho, a estrutura básica de cada tipo abstrato necessário. - Disponibilizar enunciados de problemas em que sejam necessários tipos abstratos de dados heterogêneos e solicitar que os alunos construam algoritmos para efetuar operações de entrada e saída nesses dados. O aluno deverá ser capaz de identificar, sozinho, a estrutura básica de cada tipo abstrato necessário. - Disponibilizar enunciados de problemas com tipos abstratos de dados pré-definidos (criados) pelo professor e solicitar que os alunos analisem a corretude e a necessidade dos tipos criados e sugiram alterações (justificadas).	- Capacidade de identificar as situações em que existe a necessidade de construir tipos abstratos de dados para a solução de um problema. - Capacidade de modelar os tipos abstratos necessários ao contexto de um problema específico. - Capacidade de analisar a corretude ou necessidade de tipos abstratos de dados previamente propostos por outros indivíduos para a solução de um problema específico.

Em algumas ocasiões, o programador pode se deparar com situações em que nenhuma informação pré-definida lhe é conferida. Ao invés, pode ser necessário realizar uma análise minuciosa do funcionamento de um sistema no intuito de identificar as abstrações de dados ideais e suas regras de processamento. Programadores aptos a realizar essas atividades se encontram na fase final de conhecimento denominada “Prático, Faço e Crio”. A tabela 6.16 descreve o que é necessário para que o aluno transpasse essa fase.

**Tabela 6.16 – Objetivos e Habilidades da Fase “Prático, Faço e Crio” para o Objetivo Específico
“Construir Tipos Abstratos de Dados”**

Como Encaminhar o Aluno à Síntese e Avaliação?	Como Levar o Aluno à Organização e Caracterização por um Valor ou Complexo de Valores?	Habilidade(s) a Ser(em) Adquirida(s)
- Demonstrar, a partir de um simples enunciado de problema, como levantar informações sobre um domínio (sistema), modelar os tipos abstratos de dados necessários e construir o algoritmo que solucione o problema proposto no enunciado. - Demonstrar estruturas de dados construídas para problemas específicos e como é possível otimizá-las em relação ao consumo de recurso de memória (Por exemplo, descrever um tipo de dado construído para armazenar um CPF que possui o tamanho de “15” caracteres. Em seguida, demonstrar que o CPF necessita de apenas “11” caracteres para ser armazenado, embora sua representação em tela aparente o tamanho de 14 caracteres – “999.999.999-99”).	- Disponibilizar enunciados de problemas (para domínios específicos como, por exemplo: “Programa para um sistema de controle de Vídeo Locadora”) que exigem construção de tipos abstratos de dados e solicitar que os alunos construam, além dos tipos necessários, os algoritmos para manipulação destes dados de acordo com as funções dispostas nos enunciados. - Disponibilizar algoritmos prontos para solucionar problemas previamente definidos e solicitar que os alunos avaliem a corretude dos tipos de dados homogêneos encontrados. A avaliação deve ser justificada e as estruturas de dados consideradas incorretas devem ser reescritas corretamente. - Disponibilizar algoritmos prontos para solucionar problemas previamente definidos e solicitar que os alunos avaliem a corretude dos tipos de dados heterogêneos encontrados. A avaliação deve ser justificada e as estruturas de dados consideradas incorretas devem ser reescritas corretamente. - Disponibilizar enunciados de problemas específicos com estruturas de dados já definidas e solicitar que os alunos reescrevam as estruturas no sentido de otimizá-las.	- Capacidade de construir modelos de tipos abstratos de dados para problemas ainda não vistos em sala de aula. - Capacidade de avaliar a corretude de estruturas de dados (homogêneas e/ou heterogêneas) propostas para solução de um algoritmo. - Capacidade de avaliar a possibilidade de otimização de estruturas de dados homogêneas e heterogêneas já construídas. - Capacidade de construir estruturas de dados homogêneas e heterogêneas otimizadas.

Exemplos de exercícios para este conteúdo programático podem ser encontrados no Apêndice A.5.

6.1.6 Conhecer os Tipos de Estruturas de Controle e Entender as Estruturas de Sequência

Antes de escrever um algoritmo para a solução de um problema em particular, é essencial que o programador possua uma compreensão profunda do problema e utilize uma abordagem cuidadosamente planejada para resolvê-lo (DEITEL E DEITEL, 2001). Esta abordagem planejada é influenciada pelo conhecimento do programador a respeito dos recursos (instruções e suas combinações) disponíveis em um paradigma de programação. Como exemplo, ao analisar o paradigma estruturado (ou imperativo) de programação percebem-se três tipos de instruções que regem o fluxo de execução de um algoritmo (o tipo sequencial, o condicional e o de repetição). Assim, apenas esses três tipos de instruções (em termos de fluxo de execução) podem ser utilizados para se construir programas estruturados. Através da combinação de instruções desses três tipos, é possível desenvolver algoritmos para a maioria dos problemas passíveis de solução computacional.

O primeiro tipo de instrução (o sequencial) configura-se como o mais comum de todos (inclusive, todos os algoritmos construídos pelos alunos até o momento usaram, exclusivamente, instruções do tipo sequencial). As instruções desse tipo são executadas sequencialmente na ordem em que foram escritas no algoritmo, enquanto as instruções condicionais e de repetição permitem alterar o fluxo de controle sequencial para outro fluxo determinado de acordo com a instrução utilizada.

Até então foram propostos diversos exercícios práticos para a solução de problemas simples. É importante destacar que o nível de complexidade dos algoritmos solicitados irá aumentar consideravelmente deste ponto em diante. Por este motivo, definiu-se este como o momento ideal para introdução da técnica de Programação em Duplas. É importante que o professor defina as regras para o trabalho em duplas que será implantado na disciplina. Os alunos devem ler artigos propostos pelo professor sobre Programação em Duplas e o texto “Como Agir Durante o Emparelhamento de Programadores?” – disponível no Apêndice C.

A presente seção descreve o que deve ser feito para que o aluno transpasse a fase inicial (“Vejo, Escuto e Compreendo”), intermediária (“Discuto, Aplico e Reflito”) e final

(“Prático, Faço e Crio”) do conteúdo programático “Conhecer os Tipos de Estruturas de Controle e Entender as Estruturas de Sequência” (Tabelas 5.17, 5.18 e 5.19 respectivamente).

Tabela 6.17 – Objetivos e Habilidades da Fase “Vejo, Escuto e Compreendo” para o Objetivo Específico “Conhecer os Tipos de Estruturas de Controle e Entender as Estruturas de Sequência”

Como Encaminhar o Aluno ao Conhecimento e Compreensão?	Como Levar o Aluno ao Acolhimento?	Habilidade(s) a Ser(em) Adquirida(s)
- Descrever os conceitos de estruturas de controle. - Descrever os conceitos de estruturas de controle sequenciais. - Descrever o conceito de blocos de instruções (escopo). - Descrever os conceitos de fluxograma e seus símbolos. - Demonstrar a construção de fluxogramas sequenciais.	- Solicitar que os alunos (em duplas) discutam e descrevam o máximo de instruções já utilizadas que são categorizadas como estruturas sequenciais. - Solicitar que os alunos (em duplas) construam algoritmos como solução de uma lista de problemas (como, por exemplo, problemas de matemática). - Solicitar que os alunos (em duplas) construam fluxogramas para uma lista de enunciados de problemas.	- Capacidade de relacionar o tipo de estrutura sequencial com todas as instruções já vistas até o presente momento. - Capacidade de compreender que existem situações que irão exigir mudança no fluxo de controle de execução do algoritmo. - Capacidade de compreender e construir fluxogramas.

Tabela 6.18 – Objetivos e Habilidades da Fase “Discuto, Aplico e Reflito” para o Objetivo Específico “Conhecer os Tipos de Estruturas de Controle e Entender as Estruturas de Sequência”

Como Encaminhar o Aluno à Aplicação e Análise?	Como Levar o Aluno à Resposta e Valorização?	Habilidade(s) a Ser(em) Adquirida(s)
- Descrever situações em que um bloco de instruções pode ser substituído por uma instrução ou bloco de instruções mais resumido (independente de eficiência). - Descrever macro atividades (por exemplo, atividades de leitura de dados, atividades de cálculo de uma soma, atividades de exibição de dados processados) e os blocos de instruções que comumente dão forma a elas.	- Disponibilizar uma lista com diversos enunciados de problemas (que necessitam única e exclusivamente de instruções sequenciais) e solicitar que os alunos (em duplas) construam algoritmos para resolvê-los. - Disponibilizar algoritmos prontos para a solução de problemas específicos e solicitar que os alunos (em duplas) identifiquem os possíveis escopos (blocos de instruções dependentes ou complementares). - Disponibilizar algoritmos semiprontos para a solução de problemas específicos e solicitar que os alunos (em duplas) os completem de acordo com o enunciado de cada problema.	- Capacidade de identificar blocos de instruções que são utilizados na maioria dos algoritmos (por exemplo, instrução de escrita na tela solicitando digitação seguida de instrução de leitura). - Capacidade de separar um algoritmo em diversos blocos de instruções que formam macro atividades distintas (por exemplo, blocos de instruções que culminam na entrada de dados, blocos de instruções que culminam em algum processamento matemático nos dados, blocos de instruções que culminam em saída de dados, etc.). - Capacidade de identificar o

		conjunto de instruções que devem formar um escopo específico para resolver uma determinada tarefa.
--	--	--

Tabela 6.19 – Objetivos e Habilidades da Fase “Prático, Faço e Crio” para o Objetivo Específico “Conhecer os Tipos de Estruturas de Controle e Entender as Estruturas de Sequência”

Como Encaminhar o Aluno à Síntese e Avaliação?	Como Levar o Aluno à Organização e Caracterização por um Valor ou Complexo de Valores?	Habilidade(s) a Ser(em) Adquirida(s)
- Comparar blocos de instruções com outros blocos equivalentes. - Descrever situações em que um bloco de instruções pode ser substituído por uma instrução ou bloco de instruções mais eficientes.	- Solicitar que os alunos (em duplas) desenvolvam algoritmos para problemas mais complexos não demonstrados em sala de aula até o presente momento. - Disponibilizar algoritmos prontos e solicitar que os alunos (em duplas) os avaliem em termos de eficiência e sugiram melhorias (quando possível). - Disponibilizar algoritmos prontos para a solução de um mesmo problema e solicitar que os alunos (em duplas) elaborem um resumo de comparação entre os algoritmos em termos de eficiência de código.	- Capacidade de avaliar que conjuntos de instruções são mais eficientes que outros. - Capacidade de propor melhorias em conjuntos de instruções que formam macro atividades. - Capacidade de desenvolver algoritmos para problemas ainda não demonstrados em sala de aula.

Exemplos de exercícios para este conteúdo programático podem ser encontrados no Apêndice A.6.

6.1.7 Entender as Estruturas de Seleção (Alternativas)

Conforme dito na seção anterior, estruturas de seleção são instruções que permitem alteração de fluxo de execução nos algoritmos. Essas estruturas são muito utilizadas porque, na maioria das situações (ou problemas) é necessário algum tipo de decisão que irá alterar as sequências de instruções a serem processadas. Uma estrutura de seleção permite a escolha de um grupo de ações (bloco de instruções) a ser executado quando determinadas condições,

representadas por expressões lógicas ou relacionais, são satisfeitas ou não (FORBELLONE e EBERSPACHER, 2000).

A tabela 6.20 mostra como o aluno deve tomar conhecimento e compreender os conceitos sobre as estruturas de seleção, também conhecidas como estruturas condicionais ou alternativas.

Tabela 6.20 – Objetivos e Habilidades da Fase “Vejo, Escuto e Compreendo” para o Objetivo Específico “Entender as Estruturas de Seleção (Alternativas)”

Como Encaminhar o Aluno ao Conhecimento e Compreensão?	Como Levar o Aluno ao Acolhimento?	Habilidade(s) a Ser(em) Adquirida(s)
- Descrever os conceitos de estruturas de seleção simples, composta, encadeada homogênea, encadeada heterogênea e de múltipla escolha. - Demonstrar exemplos do uso de estruturas de seleção simples, composta, encadeada homogênea, encadeada heterogênea e de múltipla escolha.	- Solicitar que os alunos (em duplas) discutam e descrevam diversos exemplos de instruções de seleção simples, composta, encadeada homogênea, encadeada heterogênea e de múltipla escolha.	- Capacidade de construir instruções de seleção simples, composta, encadeada homogênea, encadeada heterogênea e de múltipla escolha (quando as condições necessárias já estiverem pré-definidas no enunciado do problema).

O conhecimento sobre as instruções de seleção se faz primordial, entretanto, não agrega muito valor se o programador não estiver capacitado a reconhecer as situações em que o mesmo é necessário e a construir algoritmos a partir de sua aplicação. Este conhecimento pode ser considerado adquirido quando o aprendiz transpassa a fase “Discuto, Aplico e Reflito” do presente conteúdo programático (Tabela 6.21).

Tabela 6.21 – Objetivos e Habilidades da Fase “Discuto, Aplico e Reflito” para o Objetivo Específico “Entender as Estruturas de Seleção (Alternativas)”

Como Encaminhar o Aluno à Aplicação e Análise?	Como Levar o Aluno à Resposta e Valorização?	Habilidade(s) a Ser(em) Adquirida(s)
- Descrever situações em que é necessário: o uso de estruturas de seleção simples; o uso de estruturas de seleção composta; o uso de estruturas de seleção encadeada homogênea; e o uso de estruturas de seleção encadeada heterogênea. - Descrever situações em que é	- Disponibilizar uma lista de problemas em que o uso de estruturas (dos diversos tipos) de seleção é necessário e solicitar que os alunos (em duplas) desenvolvam algoritmos para solucioná-los. - Disponibilizar algoritmos incorretos com estruturas de seleção	- Capacidade de identificar situações em que é necessário o uso de estruturas de seleção simples e/ou composta. - Capacidade de identificar situações em que é necessário o uso de estruturas de seleção encadeada homogênea e/ou

necessário o uso de estruturas de seleção de múltipla escolha. - Demonstrar essas situações com exemplos práticos em que as condições estejam implícitas nos enunciados dos problemas, exigindo sua interpretação no intuito de identificar as alternativas existentes (necessárias ou complementares).	simples, composta, encadeada homogênea, encadeada heterogênea e de múltipla escolha e solicitar que os alunos (em duplas) identifiquem e corrijam os erros. - Disponibilizar algoritmos semiprontos (faltando apenas a implementação das instruções de seleção) e solicitar que os alunos (em dupla) os completem de acordo com o enunciado do problema de cada algoritmo.	heterogênea. - Capacidade de identificar situações em que é necessário o uso de estruturas de seleção de múltipla escolha. - Capacidade de construir algoritmos com o apoio de estruturas de seleção a partir da interpretação de enunciados no intuito de identificar as alternativas existentes (necessárias ou complementares).
--	---	--

As instruções de seleção podem impactar na eficiência e performance dos algoritmos. Portanto, é interessante que programadores desenvolvam a habilidade de otimizar algoritmos diminuindo a quantidade de uso dessas instruções ou reestruturando-as, objetivando-se torná-las mais eficientes (diminuindo o tempo de processamento). Considera-se que os aprendizes adquirem esta habilidade quando transpassam a fase “Prático, Faço e Crio” do presente conteúdo programático (Tabela 6.22).

Tabela 6.22 – Objetivos e Habilidades da Fase “Prático, Faço e Crio” para o Objetivo Específico “Entender as Estruturas de Seleção (Alternativas)”

Como Encaminhar o Aluno à Síntese e Avaliação?	Como Levar o Aluno à Organização e Caracterização por um Valor ou Complexo de Valores?	Habilidade(s) a Ser(em) Adquirida(s)
- Descrever situações em que é possível otimizar o uso de estruturas de seleção simples. - Descrever situações em que é possível otimizar o uso de estruturas de seleção composta. - Descrever situações em que é possível otimizar o uso de estruturas de seleção encadeada homogênea. - Descrever situações em que é possível otimizar o uso de estruturas de seleção encadeada heterogênea. - Descrever situações em que é possível otimizar o uso de estruturas de seleção de múltipla escolha.	- Disponibilizar algoritmos com estruturas de seleção de todos os tipos e solicitar que os alunos (em duplas) identifiquem melhorias e reescrevam cada algoritmo no intuito de otimizá-los.	- Capacidade de produzir algoritmos menores e mais eficientes usando estruturas de seleção simples otimizadas. - Capacidade de produzir algoritmos menores e mais eficientes usando estruturas de seleção composta otimizadas. - Capacidade de produzir algoritmos menores e mais eficientes usando estruturas de seleção encadeada homogênea otimizadas. - Capacidade de produzir algoritmos menores e mais eficientes usando estruturas de seleção encadeada heterogênea otimizadas. - Capacidade de produzir algoritmos menores e mais

		eficientes usando estruturas de seleção de múltipla escolha otimizadas. - Capacidade de identificar problemas em algoritmos construídos com o auxílio de estruturas de seleção.
--	--	--

Exemplos de exercícios para este conteúdo programático podem ser encontrados no Apêndice A.7.

6.1.8 Entender as Estruturas de Repetição (Laços)

Existem situações em que algumas instruções (blocos) devem ser executadas mais de uma vez durante o processamento de um algoritmo. Nessas situações, podem ser utilizadas estruturas de repetição (instruções que forçam a execução de blocos de instruções enquanto uma condição for verdadeira – “repetir enquanto” e “repetir para” – ou até que esta condição seja verdadeira – “repetir até que”). As estruturas de repetição permitem a construção de algoritmos mais complexos, entretanto menores.

Conhecer os conceitos das estruturas de repetição é primordial para a construção de algoritmos estruturados e capazes de resolver problemas mais generalizados como, por exemplo: “algoritmo para calcular a média das notas dos alunos de uma turma, em uma disciplina, independente da quantidade de alunos nela matriculados”. A tabela 6.23 mostra o que fazer para que os alunos transpassem a fase “Vejo, Escuto e Compreendo” do conteúdo programático “Entender as Estruturas de repetição (Laços)”.

Tabela 6.23 – Objetivos e Habilidades da Fase “Vejo, Escuto e Compreendo” para o Objetivo Específico “Entender as Estruturas de Repetição (Laços)”

Como Encaminhar o Aluno ao Conhecimento e Compreensão?	Como Levar o Aluno ao Acolhimento?	Habilidade(s) a Ser(em) Adquirida(s)
- Descrever os conceitos de estrutura de repetição com teste no início (“repetir enquanto”).	- Disponibilizar uma lista de enunciados de problemas e solicitar que os alunos (em dupla) desenvolvam algoritmos usando	- Capacidade de construir algoritmos usando laços com teste no início.

- Descrever os conceitos de estrutura de repetição com teste programado (“repetir para”). - Descrever os conceitos de estrutura de repetição com teste no final (“repetir até que”). - Demonstrar a construção de laços infinitos.	- estrutura de repetição com teste no início. - Solicitar que os alunos desenvolvam (em dupla) os mesmos algoritmos da lista anterior usando estrutura de repetição com teste programado. - Solicitar que os alunos desenvolvam (em dupla) os mesmos algoritmos da lista anterior usando estrutura de repetição com teste no final.	- Capacidade de construir algoritmos usando laços com teste programado. - Capacidade de construir algoritmos usando laços com teste no final.
--	---	--

A partir de alguns arranjos nos parâmetros de controle, os tipos de laços existentes podem ser usados em qualquer situação em que estruturas de repetição são necessárias. Contudo, a sintaxe padrão de cada um dos tipos os direciona para uma situação específica. Por exemplo: laços com teste no início são comumente utilizados em situações em que não se conhece a quantidade de vezes que o bloco de instruções interno ao laço deverá ser executado; laços com teste programado são comumente usados em situações em que se conhece a quantidade exata de vezes que o bloco de instruções interno ao laço deverá ser executado; e laços com teste no final são comumente usados em situações em que se deseja executar o bloco de instruções interno ao laço pelo menos uma vez, mas não se conhece a quantidade de vezes que ele será executado.

Conhecer e saber identificar as situações em que cada um dos tipos de estruturas de repetição é mais indicado pode ajudar na construção de algoritmos mais legíveis. Este conhecimento é adquirido quando o aprendiz transpassa a fase “Discuto, Aplico e Reflito” do presente conteúdo programático (Tabela 6.24).

Tabela 6.24 – Objetivos e Habilidades da Fase “Discuto, Aplico e Reflito” para o Objetivo Específico “Entender as Estruturas de Repetição (Laços)”

Como Encaminhar o Aluno à Aplicação e Análise?	Como Levar o Aluno à Resposta e Valorização?	Habilidade(s) a Ser(em) Adquirida(s)
- Descrever as situações em que a estrutura de repetição com teste no início é recomendada. - Descrever as situações em que a estrutura de repetição com teste programado é recomendada. - Descrever as situações em que a estrutura de repetição com teste no	- Disponibilizar uma lista de enunciados de problemas e solicitar que os alunos (em dupla) discutam e identifiquem o tipo de estrutura de repetição recomendado para a construção da solução algorítmica de cada enunciado. - Disponibilizar algoritmos	- Capacidade de identificar as situações em que a estrutura de repetição com teste no início é indicada á construção de um algoritmo específico. - Capacidade de identificar as situações em que a estrutura de repetição com teste

final é recomendada. - Demonstrar algoritmos que possuem estrutura de repetição com teste no início. - Demonstrar algoritmos que possuem estrutura de repetição com teste programado. - Demonstrar algoritmos que possuem estrutura de repetição com teste no final.	semiprontos de enunciados pré-definidos (faltando apenas a construção do laço, mas com os blocos de instrução internos à ele já incluídos) e solicitar que os alunos (em dupla) discutam e identifiquem o local em que deverá ser inserida uma estrutura de repetição adequada e completem o algoritmo.	programado é indicada a construção de um algoritmo específico. - Capacidade de identificar as situações em que a estrutura de repetição com teste no final é indicada a construção de um algoritmo específico. - Capacidade de analisar um algoritmo e identificar locais em que existe a necessidade da construção de laços e que tipos de laços são, naquele caso, mais indicados.
---	---	--

Na maioria das situações, laços são construídos de maneira aleatória. Entretanto existem casos em que uma análise minuciosa da lógica do problema pode levar os programadores a decidirem pela construção de laços de maneira otimizada. Imagine o enunciado de um problema para mostrar os números pares entre 0 e 100. Na construção do algoritmo para solucionar o problema, um laço pode ser construído aleatoriamente da seguinte maneira: “faça para (contador = 0; contador <= 100; contador = contador + 1)”. Interno ao laço também será necessário o uso de uma instrução que identifica os números pares para mostrá-los na tela. Programadores experientes são capazes de identificar que o laço não foi construído de maneira otimizada, pois está sendo executado 101 vezes. Para o problema em questão, é fácil notar que o laço ideal deve ser executado 51 vezes (pois o incremento do laço pode ser realizado de 2 em 2, uma vez que ao problema interessam apenas os números pares). Não obstante, a identificação dos números pares, interna ao laço, também se torna desnecessária (uma vez que todo número par somado ao número 2 sempre culmina em um novo número par). Assim, a sintaxe otimizada do laço é a seguinte: “faça para (contador = 0; contador <= 100; contador = contador + 2)”.

A tabela 6.25 mostra o que pode ser feito para que os aprendizes se tornem capazes de avaliar e otimizar laços de acordo com o contexto em que estão inseridos (transpassando a fase “Prático, Faço e Crio” do presente conteúdo programático).

Tabela 6.25 – Objetivos e Habilidades da Fase “Prático, Faço e Crio” para o Objetivo Específico “Entender as Estruturas de Repetição (Laços)”

Como Encaminhar o Aluno à Síntese e Avaliação?	Como Levar o Aluno à Organização e Caracterização por um Valor ou Complexo de Valores?	Habilidade(s) a Ser(em) Adquirida(s)
- Descrever como construir estruturas de repetição otimizadas. - Demonstrar como usar chavesinho (tabela de testes) para verificar a corretude de um algoritmo com laços. - Demonstrar blocos de código com estruturas de repetição para problemas específicos e otimizá-los no quadro branco. Comparar cada bloco de código original com o seu correspondente otimizado.	- Disponibilizar algoritmos com laços prontos e solicitar que os alunos (em dupla) avaliem a sua corretude de acordo com o seu enunciado (justificando os considerados incorretos e propondo as correções necessárias). - Disponibilizar algoritmos construídos com laços incorretos e solicitar que os alunos (em dupla) identifiquem e corrijam os erros. - Disponibilizar algoritmos equivalentes com laços diferentes e solicitar que os alunos (em dupla) identifiquem o algoritmo mais eficiente e descrevam o motivo. - Disponibilizar enunciados de problemas em que é necessária a construção de algoritmos que fazem uso de laços aninhados. - Solicitar que os alunos (em dupla) otimizem laços já construídos.	- Capacidade de identificar erros de lógica em laços. - Capacidade de corrigir erros de lógica identificados em laços. - Capacidade de avaliar a eficiência de um laço. - Capacidade de construir algoritmos complexos que necessitam de laços aninhados. - Capacidade de prever o resultado de variáveis em algoritmos, com laços que executam n vezes, através da tabela de testes.

Exemplos de exercícios para este conteúdo programático podem ser encontrados no Apêndice A.8.

6.1.9 Entender Modularização e Construir Software Modular

Na medida em que os problemas crescem em nível de complexidade, grandes blocos de instruções são produzidos para solucioná-los. Consequentemente, a dificuldade em construir e manter esses algoritmos, devido à quantidade de instruções combinadas, é aumentada. Não obstante, a quantidade de tarefas a que os algoritmos se dispõem a resolver

(ou que são necessárias à solução de um problema) aumenta ao passo que os problemas se tornam mais complexos.

Problemas complexos podem ser simplificados quando divididos (ou decompostos) em vários subproblemas. Decompor um problema influencia diretamente na redução de sua complexidade. É importante lembrar que complexidade é sinônimo de variedade, ou seja, a quantidade de situações diferentes que um problema pode apresentar. Assim, decompor um problema culmina na simplificação de sua solução. Outra vantagem da decomposição é a identificação e geração de módulos que podem ser reutilizados para a solução de diversos outros problemas. O processo de decomposição representa o princípio filosófico “dividir para conquistar”.

Segundo Forbellone e Eberspacher (2000), é conveniente adotar um critério para orientar esse processo de decomposição formado pelas seguintes etapas:

- Dividir o problema em suas partes principais;
- Analisar a divisão obtida para garantir coerência;
- Analisar a existência de partes que ainda podem ser decompostas e, em caso positivo, retornar à etapa de decomposição inicial;
- Analisar o resultado para garantir entendimento e coerência.

Esse processo de decomposição contínua é também conhecido como Refinamentos Sucessivos (abordagem *top-down*), porque se parte de um problema complexo e abrangente que é sucessivamente dividido até resultar em problemas mais simples e específicos.

Para decompor o problema em subproblemas, o programador deve ser capaz de construir módulos (subalgoritmos) que comporão a solução como um todo. Para adquirir esta habilidade, os aprendizes devem transpassar a fase “Vejo, Escuto e Compreendo” do presente conteúdo programático (Tabela 6.26).

Tabela 6.26 – Objetivos e Habilidades da Fase “Vejo, Escuto e Compreendo” para o Objetivo Específico “Entender Modularização e Construir Software Modular”

Como Encaminhar o Aluno ao Conhecimento e Compreensão?	Como Levar o Aluno ao Acolhimento?	Habilidade(s) a Ser(em) Adquirida(s)
- Descrever os conceitos de modularização. - Demonstrar a construção de um módulo por intermédio de função e procedimento. - Descrever os conceitos de passagem e retorno de parâmetros. - Demonstrar a construção de algoritmos com chamadas a funções e procedimentos. - Descrever os conceitos de Recursividade. - Demonstrar como substituir laços por funções ou procedimentos recursivos.	- Disponibilizar uma lista de módulos (funções e procedimentos) descrevendo os parâmetros de entrada e saída necessários à sua construção e solicitar que os alunos (em dupla) produzam os algoritmos correspondentes a cada módulo. - Disponibilizar uma lista de módulos construídos com o uso de estruturas de repetição e solicitar que os alunos (em dupla) transformem cada módulo em um procedimento recursivo equivalente.	- Capacidade de construir módulos de algoritmos a partir da pré-definição, pelo professor, dos blocos de instruções que formarão cada módulo. - Capacidade de transformar um algoritmo modularizado ou não, construído com o uso de laços, em um algoritmo recursivo.

Identificar as situações em que algoritmos permitem ser modularizados pode ser considerado trivial. Porém, identificar precisamente os módulos e seus blocos de instruções correspondentes exige estudo detalhado do problema a ser resolvido. Esta habilidade é adquirida quando os aprendizes transpassam a fase “Discuto, Aplico e Reflito” do corrente conteúdo programático (Tabela 6.27).

Tabela 6.27 – Objetivos e Habilidades da Fase “Discuto, Aplico e Reflito” para o Objetivo Específico “Entender Modularização e Construir Software Modular”

Como Encaminhar o Aluno à Aplicação e Análise?	Como Levar o Aluno à Resposta e Valorização?	Habilidade(s) a Ser(em) Adquirida(s)
- Demonstrar como identificar os possíveis módulos que compõem um enunciado de problema através da técnica de refinamentos sucessivos. - Demonstrar como identificar os possíveis módulos que compõem um algoritmo já construído para o enunciado de problema através da técnica de refinamentos sucessivos. - Demonstrar as situações em que o uso de recursividade se faz obrigatório.	- Disponibilizar uma lista de enunciados de problemas e solicitar que os alunos (em dupla) discutam e identifiquem (através de refinamento) todos os módulos necessários à construção de cada algoritmo. - Solicitar que os alunos (em dupla) construam os algoritmos da lista anterior de acordo com os módulos identificados.	- Capacidade de identificar a necessidade de utilização de funções ou procedimentos recursivos. - Capacidade de identificar e construir os possíveis módulos de um algoritmo pronto (ainda não modularizado). - Capacidade de identificar e construir os possíveis módulos que compõem a solução algorítmica do enunciado de um problema.

Por fim, mais difícil que identificar precisamente os módulos e seus blocos de instruções correspondentes é a construção de módulos otimizados e concisos, exigindo estudo ainda mais minucioso do problema a ser resolvido. Este estudo é necessário porque os módulos produzidos precisam apresentar as seguintes características: (1) alta coesão – cada módulo deve resolver a menor quantidade de tarefas possível; e (2) baixo acoplamento – é ideal que o módulo esteja conectado a outros módulos pela menor quantidade possível de parâmetros para que possa ser reutilizado em outros algoritmos com nenhuma ou poucas alterações prévias em seu código. A habilidade de construir módulos com estas características é adquirida quando os aprendizes transpassam a fase “Prático, Faço e Crio” do presente conteúdo programático (Tabela 6.28).

Tabela 6.28 – Objetivos e Habilidades da Fase “Prático, Faço e Crio” para o Objetivo Específico “Entender Modularização e Construir Software Modular”

Como Encaminhar o Aluno à Síntese e Avaliação?	Como Levar o Aluno à Organização e Caracterização por um Valor ou Complexo de Valores?	Habilidade(s) a Ser(em) Adquirida(s)
- Descrever os conceitos de coesão e acoplamento. - Demonstrar módulos diferentes para a solução de um mesmo problema e compará-los em relação à coesão. - Demonstrar módulos diferentes para a solução de um mesmo problema e compará-los em relação ao acoplamento.	- Disponibilizar módulos diferentes para a solução de um único problema e solicitar que os alunos (em dupla) identifiquem os módulos mais coesos (justificando). - Disponibilizar módulos diferentes para a solução de um único problema e solicitar que os alunos (em dupla) identifiquem os módulos menos acoplados (justificando). - Disponibilizar enunciados de problemas e solicitar que os alunos desenvolvam (em dupla) algoritmos modularizados para sua solução. Alguns destes problemas devem ser passíveis de solução, única e exclusivamente, através de módulos recursivos. É importante destacar que os alunos deverão ser capazes de identificá-los, construí-los e descrever o motivo pelo qual não podem ser construídos através do uso de estruturas de repetição. - Disponibilizar algoritmos modulares prontos e solicitar que os alunos (em dupla) identifiquem situações em que os módulos podem ainda ser divididos em módulos mais	- Capacidade de identificar situações em que é possível construir uma solução algorítmica para um problema apenas com o uso de módulos recursivos. - Capacidade de avaliar a coesão de um módulo. - Capacidade de avaliar o acoplamento de um módulo. - Capacidade de construir módulo otimizado (menos acoplado e mais coeso possível).

	coesos. Solicitar que as duplas reescrevam os módulos neste sentido. - Disponibilizar algoritmos modulares prontos e solicitar que os alunos (em dupla) identifiquem situações em que os módulos podem sofrer alterações de modo a torná-los menos acoplados. Solicitar que as duplas reescrevam os módulos neste sentido.	
--	--	--

Exemplos de exercícios para este conteúdo programático podem ser encontrados no Apêndice A.9.

6.2 OBSERVAÇÕES E DICAS IMPORTANTES

Há que se destacar o motivo pelo qual foi definido que os aprendizes devem iniciar a disciplina sob o paradigma de desenvolvimento individual. Considera-se importante que os alunos novatos iniciem individualmente no intuito de construir parâmetros de comparação quando forem apresentados ao paradigma colaborativo de Programação em Duplas. Como a disciplina é essencialmente considerada complexa e inovadora, é de se esperar que os aprendizes provarão da sensação de dificuldade em resolver os problemas durante o seu curso – principalmente nesse período inicial. Durante essa fase, é interessante que seja ventilada a idéia de que o trabalho colaborativo pode ajudá-los a construir conhecimento de maneira mais fácil e agradável. Assim, espera-se que os aprendizes construam indagações constantes sobre o processo de aprendizado individual e analisem a possibilidade de trabalhar em duplas.

Desta maneira, a partir do momento em que é apresentada a técnica de Programação em Duplas aos aprendizes, pode ser percebida uma mudança brusca de atitude em relação à sua motivação. Isso ocorre por vários motivos, dentre eles: (1) muitos alunos se sentem inibidos em levantar dúvidas aos professores (sentindo-se envergonhados) – essa inibição se reduz em quase 100% quando os alunos realizam atividades em dupla, pois, em geral, a maioria das dúvidas pode ser respondida pelo parceiro ou através do conflito sócio-cognitivo

gerado durante as discussões; (2) a compreensão dos passos lógicos (não a solução pronta) que o parceiro segue para resolver os problemas ajuda na construção do próprio estilo e lógica de programação, culminando no sentimento de estar (de fato) aprendendo; (3) cria-se uma sensação de que completar a tarefa de programação é mais fácil em dupla (e realmente o é) – isso culmina em um sentimento mais elevado de realização e satisfação; entre outros.

É fundamentalmente importante, no início do período letivo, já nas primeiras atividades extraclasse, esclarecer aos aprendizes que a tarefa de programação exige habilidades diversas que só podem ser adquiridas através de muita prática. Por este motivo, os alunos devem estar conscientes que serão submetidos a uma bateria de exercícios práticos no decorrer do curso da disciplina. Contudo, deve ser frisado que compreender a solução de outros colegas ou até mesmo a demonstração de exemplos (soluções do professor) no quadro branco não é suficiente para que o aluno adquira tais habilidades. Durante experiências passadas em docência da disciplina “Programação de Computadores”, apenas alunos com postura ativa (que procuraram solucionar os problemas e/ou encontrar melhores soluções que as disponíveis) atingiram esse objetivo. É importante destacar também que, a simples exposição aos exercícios já resolvidos leva o aprendiz a uma falsa sensação de estar apto a resolver problemas semelhantes.

CAPÍTULO 7

RESULTADOS OBTIDOS NOS EXPERIMENTOS COM O MTEA

Conforme dito anteriormente, experimentos foram realizados com turmas de programação no curso de Sistemas de Informação e no curso de Engenharia de Produção de uma Universidade da iniciativa privada. Estas disciplinas são compostas semestralmente por um número máximo de 60 alunos.

Os experimentos ocorreram durante dois anos e foram organizados em dois períodos distintos de acordo com a abordagem escolhida em relação aos objetivos da pesquisa. A primeira fase corresponde ao período de experimentos nos semestres anteriores à qualificação (1º semestre de 2008), cujo objetivo principal estava direcionado à construção de um método de ensino (baseado no modelo cognitivo de programação de Lui e Chan, 2004), nomeado *Aprendizado Extremo de Programação (eXtreme Learning of Programming – XLP)*, e a comprovar que seu uso colabora para a diminuição da evasão nas disciplinas de programação, enquanto a segunda fase corresponde ao período de experimentos nos semestres posteriores à qualificação (1º e 2º semestres de 2009), cujo objetivo principal voltou-se à construção de um método de ensino (desta vez baseado na taxionomia de objetivos educacionais de Bloom e renomeado como *Método Trifásico de Ensino-Aprendizagem*) e ao estudo dos efeitos do seu uso no comportamento dos aprendizes.

7.1 PRIMEIRA FASE DE EXPERIMENTOS

A metodologia adotada nos experimentos durante esta fase englobou três enfoques estatísticos diferentes. O primeiro enfoque envolve a comparação dos dados de evasão de alunos da disciplina “Algoritmos e Técnicas de Programação” (ATP) – equivalente à disciplina “Programação de Computadores” – que trabalharam sob o método de programação tradicional com os alunos que trabalharam sob o método XLP. O segundo enfoque descreve a coleta de opiniões de uma amostra dos alunos que cursaram a disciplina sob o método XLP; e o último enfoque demonstra a análise do desenvolvimento dos alunos que participaram de um grupo de estudos extraclasse sob o método XLP.

7.1.1 Primeiro Enfoque

Para realizar a pesquisa sob o primeiro enfoque, utilizou-se a técnica de Programação em Duplas com os alunos matriculados no primeiro semestre de 2008 na disciplina ATP do curso de Sistemas de Informação. Em seguida, foi analisado o histórico das turmas anteriores (desde o ano 2000) dessa mesma disciplina a partir de dados obtidos junto à Secretaria Acadêmica do Campus. Como variável de análise, foi escolhido o número de desistências em cada semestre de oferta da disciplina. Foram considerados alunos desistentes aqueles que trancaram matrícula na disciplina, aqueles que não fizeram a avaliação global e especial, e os que foram reprovados por frequência. A partir da análise desses comportamentos, foram obtidas as proporções de desistentes desde o início do curso. Essa proporção é o resultado da razão entre o número de alunos desistentes na disciplina e o número total de alunos matriculados na mesma (em um determinado semestre letivo).

A partir das proporções de desistentes, foi observada uma diferença nos dados do primeiro semestre de 2008 em relação aos demais períodos escolares analisados, porém, procurou-se verificar a sua significância estatística.

Para efetuar os cálculos estatísticos, foram utilizadas as seguintes fórmulas:

(i). Fórmula da Proporção:

$$\rho = \frac{x}{n}$$

em que:

X: Número de desistentes na turma

n: Número total de alunos na turma

(ii). Fórmula da Proporção ponderada:

$$\hat{\rho} = \frac{(x1 + x2)}{(n1 + n2)}$$

em que:

x1: Número de desistentes da turma de 2008

x2: Número de desistentes da turma comparada

n1: Número total de alunos da turma de 2008

n2: Número total de alunos da turma comparada

(iii). Fórmula do teste de hipótese:

$$Z_{cal} = \frac{(\rho 2 - \rho 1)}{\sqrt{\hat{\rho}(1 - \hat{\rho})\left(\frac{1}{n1} + \frac{1}{n2}\right)}}$$

Os resultados obtidos através do cálculo de inferência estatística (Z_{cal}) foram comparados ao valor de aproximação normal igual a 1.65 (nível de significância igual a 0,10 $\rightarrow \alpha = 0,1$), obtido através da Tabela de Faixa Central proposta justamente para problemas de inferência estatística. Dessa forma:

Para $-1.65 \leq Z_{cal} \leq 1.65$, aceita-se H_0 (hipótese de que, com 90% de confiança, a diferença entre a proporção de desistentes de duas turmas não é considerada estatisticamente significativa).

Para $Z_{cal} < -1.65$ e $Z_{cal} > 1.65$, rejeita-se H_0 e aceita-se H_1 , (hipótese de que, com 90% de confiança, a diferença entre a proporção de desistentes de duas turmas deve ser considerada estatisticamente significativa).

Foi exigido, durante o ensino de ATP, que os alunos se tornassem capazes de realizar as seguintes atividades: produção da declaração de objetivos para a tarefa ou problema; construção do diagrama de atividades e do algoritmo; e codificação.

7.1.1.1 Resultados do Primeiro Enfoque

Os dados obtidos na Secretaria Acadêmica para o desenvolvimento do primeiro enfoque podem ser observados na Tabela 7.1.

Tabela 7.1. Histórico dos dados dos alunos na disciplina de ATP

Turma	Quantidade de Alunos	Quantidade de Desistentes	Proporção de Desistentes
1—2000	58	8	0.138
2—2000	65	12	0.19
1—1—2001	37	7	0.19
2—1—2001	40	9	0.23
1—2—2001	37	7	0.19
2—2—2001	39	15	0.39
1—1—2002	40	12	0.30
2—1—2002	43	6	0.1395
1—2—2002	26	6	0.23
2—2—2002	35	5	0.143
1—2003	41	7	0.17
2—2003	46	20	0.44
1—2004	61	30	0.49
2—2004	44	18	0.41
1—2005	50	9	0.18
2—2005	28	9	0.32
1—2006	54	8	0.15
2—2006	35	10	0.29
1—2007	33	20	0.61
1—2008	50	7	0.14

Foi definido para todas as comparações que:

- (i). $H_0: \rho_1 = \rho_2$ (Diferença não estatisticamente significativa)
(ii). $H_1: \rho_1 \neq \rho_2$ (Diferença estatisticamente significativa)

Em que ρ_1 sempre corresponde à proporção de desistentes de 2008 e ρ_2 corresponde à proporção de desistentes da turma que está sendo comparada.

Apenas o primeiro teste (realizado com a turma de 2007) pode ser visto a seguir. Os resultados dos demais testes se encontram na tabela 7.2 (adiante).

Primeiro Teste: Turma de 2008 versus turma de 2007:

ρ_1 : Proporção de desistentes na turma de 2008= 0.14

ρ_2 : Proporção de desistentes na turma de 2007= 0.61

x_1 : Número de desistentes na turma de 2008= 7

x_2 : Número de desistentes na turma de 2007= 20

n_1 : Número total de alunos na turma de 2008= 50

n_2 : Número total de alunos na turma de 2007= 33

Segundo a Formula da proporção ponderada:

$$\hat{\rho} = 0,33$$

Resolvendo a Fórmula do teste de hipótese, obteve-se:

$$Z_{cal} = 4.48$$

Conclusão do 1º teste:

Como $4,48 > 1,65$, rejeita-se com 90% de confiança a hipótese de que as proporções populacionais de desistências são similares, ou seja, afirma-se com 90% de confiança que a diferença dos dados de evasão entre as turmas comparadas é estatisticamente significativa.

Tabela 7.2. Resultados dos Testes de Hipótese

Testes	Turmas Comparadas	Número de alunos	Proporções de Desistentes	Zcal	Conclusão
1º Teste	Turma 2008 X Turma 2007	50	0.14	4.48	É estatisticamente significativo
		33	0.61		
2º Teste	Turma 2008 X Turma 2-2006	50	0.14	1.71	É estatisticamente significativo
		35	0.29		
3º Teste	Turma 2008 X Turma 1-2006	50	0.14	0.147	Não é estatisticamente significativo
		54	0.15		
4º Teste	Turma 2008 X Turma 2-2005	50	0.14	1.875	É estatisticamente significativo
		28	0.32		
5º Teste	Turma 2008 X Turma 1-2005	50	0.14	0.55	Não é estatisticamente significativo
		50	0.18		
6º Teste	Turma 2008 X Turma 2-2004	50	0.14	2.94	É estatisticamente significativo
		44	0.41		
7º Teste	Turma 2008 X Turma 1-2004	50	0.14	3.89	É estatisticamente significativo
		61	0.49		
8º Teste	Turma 2008 X Turma 2-2003	50	0.14	3.15	É estatisticamente significativo
		46	0.44		
9º Teste	Turma 2008 X Turma 1-2003	50	0.14	0.40	Não é estatisticamente significativo
		41	0.17		
10º Teste	Turma 2008 X Turma2 2-2002	50	0.14	0.04	Não é estatisticamente significativo
		35	0.143		
11º Teste	Turma 2008 X Turma1 2-2002	50	0.14	0.99	Não é estatisticamente significativo
		26	0.23		
12º Teste	Turma 2008 X Turma2 1-2002	50	0.14	-0.0069	Não é estatisticamente significativo
		43	0.1395		
13º Teste	Turma 2008 X Turma1 1-2002	50	0.14	1.86	É estatisticamente significativo
		40	0.30		
14º Teste	Turma 2008 X Turma2 2-2001	50	0.14	2.69	É estatisticamente significativo
		39	0.39		
15º Teste	Turma 2008 X Turma1 2-2001	50	0.14	0.63	Não é estatisticamente significativo
		37	0.19		
16º Teste	Turma 2008 X Turma2 1-2001	50	0.14	1.11	Não é estatisticamente significativo
		40	0.23		
17º Teste	Turma 2008 X Turma1 1-2001	50	0.14	0.625	Não é estatisticamente significativo
		37	0.19		
18º Teste	Turma 2008 X Turma 2-2000	50	0.14	0.70	Não é estatisticamente significativo
		65	0.19		
19º Teste	Turma 2008 X Turma 1-2000	50	0.14	-0.03	Não é estatisticamente significativo
		58	0.138		

Um conjunto de oito, dentre os dezenove testes realizados, foram considerados estatisticamente significativos, sendo que, em todos eles, percebeu-se menor evasão na turma de 2008 (que trabalhou sob o método de Programação em Duplas). Apenas duas turmas apresentaram menor evasão quando comparadas à turma de 2008. Contudo, essas turmas compõem os demais testes cuja diferença não foi considerada estatisticamente significativa.

7.1.1.2 Discutindo os Resultados do Primeiro Enfoque

No momento em que concluir-se-ia sobre os indícios de que o uso de Programação em Duplas contribuiu para a diminuição da evasão na disciplina ATP, surgiram os seguintes questionamentos: Seria a motivação, o único fator a influenciar na evasão? Em que grau a presença ou a falta de motivação pode influenciar na evasão?

Buscando respostas, concluiu-se que a evasão, em geral, é consequência de uma quantidade diversificada de fatores, que podem ocorrer individualmente ou concomitantemente entre si. Como nenhum questionamento foi feito aos alunos desistentes no período entre 2000 e 2008, considerou-se prudente descartar a hipótese inicial, embora empiricamente acredita-se na influência positiva da motivação no processo de ensino/aprednizado de programação.

7.1.2 Segundo Enfoque

O segundo objetivo enfocou avaliar a satisfação dos aprendizes com a utilização da técnica de emparelhamentos. Por este motivo, foi aplicado um questionário aos alunos que cursaram a disciplina ATP no primeiro semestre de 2008.

7.1.2.1 Resultados do Segundo Enfoque

Foi analisada a avaliação dos alunos quanto à técnica de Programação em Duplas. Sob esse aspecto, um percentual de 44% dos entrevistados conceituou a técnica como ótima, enquanto outros 56% disseram que a técnica é boa. Na pesquisa, ainda continham possíveis respostas como regular e péssima, mas nenhum dos entrevistados optou por uma dessas alternativas. Esses dados podem ser observados na Figura 7.1.

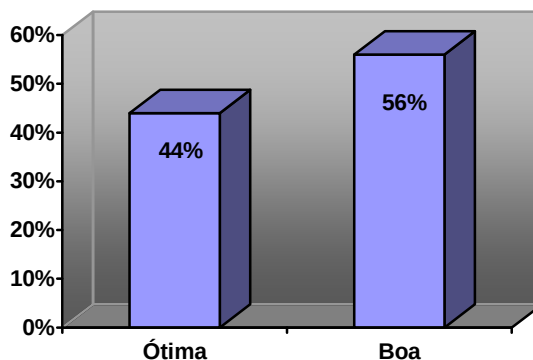


Figura 7.1. Avaliação da Programação em Duplas.

Dentre as vantagens da Programação em Duplas, mais percebidas pelos alunos, destacam-se: (1) Maior facilidade em desenvolver programas devido à ajuda do sócio (parceiro); (2) O compartilhamento de informações; (3) A interação entre as duplas; (4) Maior confiança em programar; e (5) Desenvolvimento de códigos menores.

Em seguida, os entrevistados foram questionados sobre as desvantagens observadas. A maior representatividade foi: “o parceiro pode não se interessar em colaborar com o desenvolvimento”.

Também foi pesquisado sobre a opinião dos alunos quanto à influência da Programação em Duplas no aprendizado de programação. Entre os entrevistados, 94% afirmaram que a técnica influencia no aprendizado de programação. Somente 6% responderam negativamente, como pode ser observado na Figura 7.2.

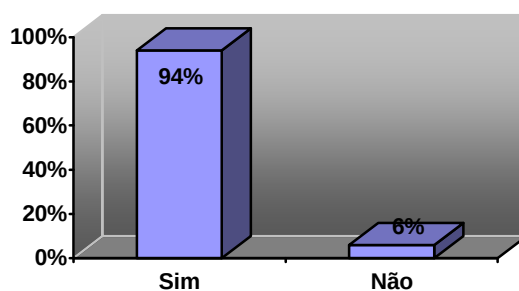


Figura 7.2. Influência da PP no Aprendizado de Programação.

Relacionada a essa pergunta, verificou-se também se a técnica de Programação em Duplas auxiliou no aprendizado da disciplina ATP. Do montante dos entrevistados, 20% afirmaram que a técnica não ajudou no seu aprendizado, enquanto 70% disseram que sim e outros 10% não opinaram. Alguns ainda justificaram sua resposta dizendo que tiveram pouco contato com a técnica. Outros, por sua vez, ao responderem “sim”, reafirmaram a sua experiência através de frases como as que se seguem:

- “... o compartilhamento de dúvidas e respostas desenvolve a capacidade de programar...”;
- “... as dúvidas que seriam tiradas com o professor foram discutidas em dupla, o que gerou muito conhecimento...”;
- “... porque trocamos informações, tiramos dúvida, o que facilita o aprendizado...”;
- “... através do meu parceiro e dos debates que realizamos, descobri como fazer determinado algoritmo, melhorando o meu desempenho nas disciplinas de programação...”.

7.1.2.2 Discutindo os Resultados do Segundo Enfoque

Pela análise dos dados, verificou-se que os alunos apreciaram a experiência de aprender com o apoio da Programação em Duplas, enquanto a maioria descreveu que a aplicação da técnica é válida no auxílio à aprendizagem em disciplinas de programação.

Esses resultados reforçaram a teoria de que a Programação em Duplas pode ser usada como recurso pedagógico no ensino da disciplina “Programação de Computadores”. Isso serviu como incentivo para prosseguir com os experimentos nos semestres seguintes (descritos na seção 7.2).

7.1.3 Terceiro Enfoque

O terceiro enfoque visou analisar o desenvolvimento dos alunos que participaram do curso extraclasse de Programação em Duplas. Para tal, foi criado um grupo de estudos no qual se inscreveram alunos de todos os períodos – totalizando 30 alunos. Os alunos desse curso foram submetidos ao método MTEA durante o período de quatro horas semanais, sendo duas horas às sextas feiras e duas horas aos sábados. Foi solicitado que os participantes desenvolvessem programas e solucionassem exercícios de lógica a partir de dinâmicas voltadas para o trabalho em dupla.

Os alunos foram treinados nos primeiros encontros com o objetivo de adquirir habilidades colaborativas. A rotatividade entre as duplas foi encorajada com a finalidade de definir quais sócios se identificavam mais, ou eram mais compatíveis entre si – produzindo as atividades de forma mais eficiente.

7.1.3.1 Resultados do Terceiro Enfoque

Para a realização da pesquisa, sob esse enfoque, observou-se o comportamento dos alunos no decorrer das aulas, bem como sua atitude, relacionamento com o sócio de dupla e a disposição para o trabalho em dupla. Através dessa observação, pôde-se destacar uma evolução quanto à interação entre os sócios. No início dos emparelhamentos, os alunos demonstraram um comportamento tímido, atenuado no decorrer do período letivo.

Percebeu-se que alguns alunos apresentaram dificuldades em se manter na sua devida posição (navegador ou piloto), pois, quando identificavam erros, procuravam corrigi-los imediatamente, apressando-se do teclado, mesmo que atuando na posição de navegador.

Os alunos do curso extraclasse de Programação em Duplas também foram submetidos a um questionário semanal que visou identificar o nível de satisfação dos alunos através dos

emparelhamentos. Dentre as perguntas feitas destaca-se: “Você gosta de programação, sim ou não?”. A Tabela 7.3 mostra os resultados obtidos.

Tabela 7.3. Índice de Satisfação dos Alunos com Programação

Pesquisa	Quantidade de Alunos Que Não Gostam de Programação	Quantidade de Alunos Que Gostam de Programação
1ª (semana)	22	8
2ª (semana)	21	9
3ª (semana)	21	9
4ª (semana)	17	13
5ª (semana)	17	13
6ª (semana)	17	13
7ª (semana)	14	16
8ª (semana)	14	16

Segundo a Tabela 7.3, o índice de alunos que apreciam programação aumentou em 100% entre a primeira e a última semana do curso. Esse resultado é muito expressivo para a pesquisa, pois demonstra que os alunos adquiriram uma atitude mais positiva em relação à construção de programas.

Em questionamentos não formalizados, percebeu-se que diversos alunos sugerem que a técnica de Programação em Duplas estimula o raciocínio porque eles são obrigados a pensar e a discutir soluções antes da escrita do código fonte e, mesmo durante a codificação, eles intervêm no trabalho do piloto, sugerindo ou buscando explicações. Essas intervenções são uma das fortes características da Programação em Duplas.

7.1.3.2 Discutindo os Resultados do Terceiro Enfoque

A Programação em Duplas foi positivamente aceita também pelos participantes do curso extraclasses. Isso pode ser observado nas transcrições (a seguir) de frases dos próprios alunos:

- “... às vezes me sinto mais importante quando minhas idéias são aceitas... isso ocorreu várias vezes durante as seções de emparelhamento...”;

- “... em alguns momentos, tivemos que nos conter enquanto navegadores para não interferir na tarefa do parceiro... isso ocorreu por motivo de ansiedade... vontade de ver o programa funcionando de fato...”;
- “... a Programação em Duplas tem ajudado na resolução de problemas simples. Nós compartilhamos nossos conhecimentos, chegando a um resultado produtivo, além de aumentar nosso desempenho e vontade de programar. O emparelhamento ajuda no ânimo de cada programador, que tende a desistir menos perante as dificuldades encontradas na programação...”.

No início do curso extraclasse, ficou notória a necessidade de aplicar treinamento com o objetivo de disseminar habilidades colaborativas entre os aprendizes. Os alunos se mostraram acostumados (como um vício) a desenvolver trabalhos usando a técnica de dividir para conquistar. Como os fundamentos da Programação em Duplas são contrários aos fundamentos da divisão e conquista (pois os alunos devem trabalhar conjuntamente na mesma atividade, e não dividi-las a fim de solucioná-las individualmente), tornou-se necessário elaborar e aplicar dinâmicas colaborativas aos aprendizes. Os alunos foram incentivados a montar objetos usando “Lego” (como, por exemplo, barcos, aviões, veículos, dentre outros) em duplas. Os grupos foram acompanhados e, na presença de todos os alunos, foram feitas intervenções elucidativas. É importante destacar que esta abordagem tornou o ambiente mais descontraído e agradável. É importante destacar que isso pode ter influenciado na aceitação da técnica de Programação em Duplas.

7.1.4 Problemas Encontrados na Primeira Fase de Experimentos

Talvez o maior problema enfrentado ao aplicar a técnica de Programação em Duplas tenha sido o fator cultural. A maioria absoluta dos aprendizes não se mostrou preparada culturalmente para desenvolver trabalhos colaborativos. Alguns motivos se destacaram, dentre eles: (1) Existe um conceito enraizado (e equivocado) entre os alunos de que trabalho colaborativo nada mais é do que a simples divisão de tarefas; (2) Raros aprendizes demonstraram (inicialmente) apreço pelo conflito sócio-cognitivo; e (3) Dificuldade por parte

de alguns alunos em aceitar sair de sua zona de conforto – trabalhar colaborativamente exige bruscas mudanças posturais como, por exemplo, preocupar-se também com o aprendizado e satisfação do próximo. Para amenizar o problema, definiu-se como fundamentalmente importante iniciar o semestre letivo aplicando dinâmicas colaborativas. Essas dinâmicas foram planejadas e elaboradas a fim de iniciar um processo de aculturação com o objetivo de tornar os aprendizes mais colaborativos.

Outro problema que se mostrou latente é a inexperiência dos aprendizes em trabalhar em duplas. O trabalho colaborativo, na perspectiva da Programação em Duplas, é guiado pelo conflito sócio-cognitivo. Administrar conflito sócio-cognitivo não compõe as diversas habilidades adquiridas pela maioria dos aprendizes durante toda a sua vida acadêmica. Compreender as regras da Programação em Duplas e as atitudes de bom senso que promovem a colaboração é primordial para que a dupla se torne pedagogicamente efetiva. Por este motivo, foi elaborado um documento guia (denominado: “COMO AGIR DURANTE O EMPARELHAMENTO DE PROGRAMADORES?”). Sua leitura foi sugerida durante todo o semestre letivo. Este documento pode ser encontrado no Apêndice C.

Percebeu-se uma forte pressão por parte dos aprendizes a fim de que as atividades avaliativas tradicionais fossem realizadas em duplas. Contudo, constantemente, procurou-se esclarecer que o objetivo da técnica de Programação em Duplas, neste contexto, envolve o desenvolvimento individual através da colaboração, e não o desenvolvimento coletivo. Esta, inclusive, foi uma das medidas tomadas para evitar a dependência entre os sócios.

7.2 SEGUNDA FASE DE EXPERIMENTOS

As seguintes turmas foram submetidas à segunda fase de experimentação: turma de “Algoritmos e Técnicas de Programação – ATP” do 1º e 2º semestres de 2009 do curso de Sistemas de Informação; e turma de “Programação de Computadores – PC” do 2º semestre de 2009 do curso de Engenharia de Produção.

Novamente, foram criadas seções de emparelhamento durante as aulas práticas de programação. Ao final de cada seção, solicitou-se que os participantes elaborassem um relatório descrevendo sua percepção do processo de emparelhamento naquele dia – os alunos foram incentivados a escrever (individualmente e sigilosamente), entre outras coisas, o que consideraram pontos positivos e/ou negativos durante os emparelhamentos.

A partir da experiência adquirida durante a primeira fase de experimentos, procurou-se aplicar a Programação em Duplas de maneira sistematizada. O processo seguiu as seguintes etapas: (1) Desenvolvimento individual de habilidades lógico matemáticas; (2) Aculturamento e treinamento dos aprendizes em habilidades colaborativas; e (3) Aprendizado de programação apoiado pelos emparelhamentos.

A primeira e segunda etapa ocorreram durante a exposição dos cinco primeiros conteúdos programáticos vistos na seção 6.1 A terceira etapa iniciou em conjunto com a explanação dos conceitos de estruturas de controle para produção de algoritmos.

Exercícios práticos para treinar a atenção e a lógica matemática (Apêndice B) foram distribuídos aos aprendizes durante a primeira etapa. A maioria deles foi resolvida no quadro branco ou em arguições orais durante as aulas teóricas iniciais.

7.2.1 Comportamentos Percebidos nas Seções de Emparelhamento Durante a Segunda Fase de Experimentos

A partir da sistematização dos experimentos, foram detectados alguns comportamentos dos alunos que são responsáveis pelo aumento da motivação e pela efetividade pedagógica do processo de emparelhamento, dentre eles: (1) pressão da dupla; (2) aumento de coragem devido ao emparelhamento; (3) aprendizado em dupla; (4) depuração em dupla; (5) revisão contínua; (6) conflito sócio-cognitivo; e (7) confiança na parceria.

7.2.1.1 Pressão da dupla

Os programadores em dupla se esforçam mais durante a tarefa de programação porque eles não querem decepcionar o seu sócio. Isto pode ser observado nas frases dos próprios alunos quando opinavam sobre seu ponto de vista após cada seção de emparelhamento:

- “... o problema de hoje estava muito difícil. Só não desisti porque não podia abandonar meu parceiro...”;
- “... eu estava com muita vergonha porque não estava conseguindo acompanhar meu parceiro. Às vezes pensei que ele poderia se chatear comigo porque quase todas as decisões foram tomadas por ele...”;
- “... eu não queria parar de fazer o programa enquanto não estivesse pronto. Mas não deu tempo de terminar. Acho que meu parceiro ficou chateado com isso, pois eu fiquei chateada...”;
- “... hoje fizemos três tarefas pequenas. Vibramos muito nas duas vezes em que terminamos as tarefas antes de todas as outras duplas... Acho que minha parceira gostou de trabalhar comigo, pois tivemos uma sintonia boa...”;
- “... hoje foi muito bom. Percebi que ajudei muito meu parceiro a desenvolver os dois programas. Estou meio aliviada porque acho que na semana passada foi ele quem fez praticamente tudo...”;
- “... cheguei atrasado por causa da minha Van. Isso me chateou porque não conseguimos fazer o programa a tempo. Meu parceiro estava achando que eu estava enrolando para chegar atrasado. Mas ele acreditou em mim, ainda bem...”.

Quando os programadores se reúnem com o sócio, ambos trabalham intensivamente porque eles são motivados a completar a tarefa durante aquela sessão de emparelhamento. Assim, as duplas tendem a valorizar mais o seu tempo compartilhado. Eles não conferem mensagens de e-mail constantemente, evitam entrar no MSN, ORKUT e outras diversões na

Internet. Eles não querem desperdiçar o tempo do outro. Exemplos podem ser vistos nas frases a seguir:

- “... não deu nem pra ler meu e-mail hoje. Mas está bom, pelo menos terminamos o programa na aula...”;
- “... precisava ir à biblioteca pegar um livro, mas não tive coragem. Estava tão ansiosa para ver o programa funcionando que deixei a biblioteca para depois...”;
- “... fui entrar no MSN hoje, na máquina do lado, e minha parceira não deixou. Fiquei numa raiva... mas depois vi que eu estava errada. Não faço mais isso...”.

Quando se trabalha só, é possível vagar pelo dia. É possível entrar à revelia naquele estado mental chamado de “viagem”; É possível navegar à vontade na Internet; É possível se dirigir às reuniões e almoçar quando desejado. Tudo é muito diferente quando se trabalha emparelhado. É preciso coordenar horários. Não obstante, este poderia ser um motivo em potencial pelo qual algumas pessoas poderiam não gostar de Programação em Duplas. Contudo, quando o objetivo da Programação em Duplas é acadêmico, espera-se que os participantes (neste caso, aprendizes) da seção de emparelhamento se encontram ali por livre e espontânea vontade e previamente se dirigem à aula prática conscientes do seu ofício. A maioria dos alunos sentiu o peso desta responsabilidade. Indícios podem ser vistos nas frases a seguir:

- “... outros colegas nos vêm trabalhando com nosso parceiro e nos deixam sós. Isso é bom porque temos uma maior quantidade de tempo ininterrupto, que é bom para manter-nos focados a terminar o programa...”;
- “... eu precisava conversar com a... mas não tive coragem. Não quis deixar meu parceiro sozinho e ainda fiquei com medo do parceiro dela não gostar...”;
- “... eu precisava matar aula para estudar para a prova de... mas se fizesse isso, eu estaria abandonando meu parceiro. Se ele fizesse isso comigo, eu me chatearia...”;
- “... puxa vida! Parece que não deu tempo nem de respirar hoje. Que tarefa difícil. Ainda bem que meu parceiro foi paciente, porque me deu vontade de desistir várias vezes...”.

Por fim, outro benefício da pressão da dupla foi percebida nos códigos resultantes em forma de aderência a procedimentos e padrões impostos pelo professor. Um exemplo disso é a quantidade de comentários bem elaborados percebida na maioria dos programas produzidos pelas duplas. Outro exemplo expressivo é a alta quantidade de “constantes” encontradas nestes programas.

7.2.1.2 Conflito sócio-cognitivo

Percebeu-se que o conflito sócio-cognitivo trouxe certa melhoria de qualidade quando analisadas as soluções das tarefas de programação. Alguns motivos se destacam:

- Cada sócio traz, à dupla, as suas habilidades e perspectivas;
- Ambos compartilham da mesma meta de completar a tarefa;
- Os sócios abordam um problema conjuntamente, cada qual com uma sugestão de alternativa para resolvê-lo;
- Os pares têm que negociar sobre como abordar o problema. Nesta negociação, eles avaliam mais alternativas que qualquer um dos dois teria considerado sozinho. Uma pessoa que trabalha só tende a procurar e seguir a primeira abordagem que vem à mente;
- Em conjunto, os sócios determinam eficazmente qual é o melhor plano de ataque, enquanto considerando e incluindo sugestões de ambos programadores.

7.2.1.3 Aumento de coragem devido ao emparelhamento

Percebeu-se que os alunos em dupla demonstraram mais coragem para apresentar os resultados dos seus trabalhos no quadro branco. Isso ocorreu porque eles adquiriram mais confiança nas próprias soluções. Essa coragem é interessante, pois transforma a aula em um rico ambiente participativo – os alunos aprendem muito quando assistem outros colegas demonstrando suas soluções no quadro branco. Exemplos de coragem podem ser vistos nas frases a seguir:

- “... quando nós trabalharmos com outra pessoa, nós criamos bastante conhecimento para nos sentir mais confiantes no que estamos fazendo...”;
- “... com o meu parceiro, eu tive mais coragem de perguntar e admitir quando não sabia algo...”;
- “... engraçado, mas agora acho que estou entendendo mesmo a matéria...”;
- “... eu nunca sou convidado a colocar no quadro a nossa solução... eu terei a minha vez?...”;
- “... eu achei que morreria de vergonha quando tivesse que ir ao quadro, mas fiquei tranquila, pois sabia que nossa solução estava correta...”.

7.2.1.4 Revisão contínua

As duplas conseguiram desenvolver programas com projetos mais refinados e menos falhos que os mesmos programas de outros alunos de turmas anteriores que trabalharam individualmente. Isso ocorreu por causa do processo de revisão contínua a que é submetido o código durante os emparelhamentos. Conforme observado, o aluno na posição de navegador, na maioria absoluta das vezes, encontrou erros gramaticais enquanto o piloto digitava o

código e, em diversas outras situações, encontrou também erros semânticos, conhecidos como erros de lógica.

7.2.1.5 Depuração em dupla

É possível que todo indivíduo já tenha experimentado problemas que simplesmente podem ser solucionados pelo ato de explicá-los a outra pessoa. Isso ocorre porque o ouvinte normalmente questiona, fazendo com que o interlocutor descreva alguns detalhes da sua lógica que antes permaneciam implícitos. Ficou notório como essas perguntas e respostas podem conduzir a grandes revelações. Frases dos alunos (a seguir) mostram como este comportamento ocorreu em diversas situações:

- “... é engraçado, mas, algumas vezes eu encontrei erros na minha lógica quando me dispus a explicar ao meu parceiro como eu estava pensando...”;
- “... hoje eu vi que estava pensando errado um monte de vezes... isso ocorreu na maioria das vezes em que eu descrevia minhas idéias ao meu parceiro...”;
- “... meu parceiro é muito bom de lógica. Toda vez que tento lhe explicar algo, ele me faz umas perguntas que colocam minhas idéias por terra...”;
- “... fiz uma pergunta para minha parceira sobre um aspecto do programa e me veio à mente uma resposta que simplesmente resolveu todo o problema. Não deu nem tempo dela pensar...”.
- “... hoje minha parceira me fez uma pergunta que me ajudou a matar a charada do programa naquele instante...”;

7.2.1.6 Aprendizagem em dupla

O conhecimento é constantemente compartilhado entre os sócios. Esse conhecimento pode estar relacionado ao raciocínio lógico, estilo de programação e até mesmo conhecimento de sintaxe de linguagem. Os alunos se tornam, por diversas vezes, professores dos seus sócios. Isso é interessante do seguinte ponto de vista: Embora não se buscou fundamentação científica para a seguinte afirmação, é conhecimento empírico afirmar que, quando se empenha em ensinar algo a alguém, desenvolve-se um raciocínio mais sistematizado e minucioso sobre o objeto de estudo para se fazer compreendido por esta pessoa. Deste ponto de vista, portanto, desenvolver programas em dupla é valioso para quem aprende (pois é agraciado com a proeminência de um professor particular, exclusivamente dedicado a ele) e para quem ensina (pois, quem ensina termina por aprender conceitos em um nível mais elevado de conhecimento).

É interessante também perceber que os alunos menos experientes, durante o processo de emparelhamento, normalmente adquirem habilidades explícitas de ouvir, de ver os alunos mais experientes construírem suas conexões lógicas para resolver um determinado problema e de analisar o produto que está sendo construído sob um ponto de vista crítico.

As frases seguintes reforçam essa teoria:

- “... aprendi muito mais hoje com meu parceiro do que com a explicação do professor...”;
- “... tudo o que eu não entendo, quando o professor explica, eu acabo aprendendo com meu parceiro... isso é normal? O que eu posso fazer para compreender melhor quando o professor explica?...”;
- “... parece um jogo de paciência. Às vezes meu parceiro perde uns 20 minutos para me explicar algo. Porém, percebo que a explicação dele me coloca dentro do contexto e, assim, consigo participar melhor da produção do programa...”;
- “... hoje me senti importante. Expliquei tudo para o meu parceiro e conseguimos desenvolver o programa quase que somente com as minhas idéias...”;

- “... parece que a gente aprende mais quando está ensinando. Isso é engraçado e ao mesmo tempo estranho...”;
- “... hoje eu aprendi um monte de comandos em C com minha parceira. Tive a sensação de que estou boiando nas aulas teóricas, pois ela mostrou que conhece diversas instruções que eu nem tinha idéia...”.

7.2.1.7 Confiança na parceria

Pares também têm que confiar um no outro. Nos casos em que existiu sinergia entre os sócios, esta confiança surgiu naturalmente. Emparelhar os aprendizes, na maioria das duplas, permitiu a troca de experiência de modo tal que eles mesmos foram capazes de perceber que o resultado alcançado em dupla foi melhor do que poderia ter sido se eles tivessem desenvolvido os trabalhos individualmente. Isso também pode ser percebido (a seguir) nas próprias observações deles:

- “... hoje nós fizemos um programa que, sinceramente, eu não faria sozinha nunca...”;
- “... fizemos um programa difícil hoje. Ficamos nos perguntando se conseguiríamos fazê-lo sozinhos...”;
- “... a gente tem a sensação de que não dá para fazer alguns programas sozinhos...”.

7.2.2 Problemas Percebidos nas Seções de Emparelhamento Durante a Segunda Fase de Experimentos

Seria muito bom se tudo sobre Programação em Duplas fosse perfeito, ou se a técnica fosse uma cura milagrosa para todos os problemas de aprendizado de programação. Porém, existem alguns problemas iminentes e precauções a serem tomadas para que a Programação em Duplas seja tão efetiva quanto possível.

7.2.2.1 A dependência

É possível que alguns aprendizes desenvolvam uma não aconselhável dependência do seu sócio. Embora, para muitos, seja verdadeiramente prazeroso trabalhar em duplas, é importante que o aprendiz se sinta também capaz de desenvolver seus trabalhos individualmente. Percebeu-se esse risco prematuramente e definiu-se que somente 10% da pontuação semestral seria alocada a atividades em duplas. Isso contribuiu para uma consciência geral de que a Programação em Duplas estava sendo experimentada no intuito de melhorar o aprendizado individual, e que o aprendizado coletivo seria apenas considerado consequente.

7.2.2.2 A falta da rotatividade

Muitos aprendizes se acostumam com seus sócios a ponto de não aceitarem outros colegas como sócios. Isso pode ocorrer por um entre vários fatores: (1) O aprendiz se tornou dependente do sócio; ou (2) O aprendiz se tornou complacente e deixa o sócio executar a tarefa de programação sempre; ou (3) O aprendiz construiu um sentimento de afinidade muito grande com o sócio, a ponto de acreditar que não será tão produtivo com outros sócios.

Nenhum destes motivos é benéfico ao processo de aprendizado. Embora a afinidade possa ser considerada por diversas vezes favorável ao conflito sócio-cognitivo, é importante destacar que a rotatividade incentiva o compartilhamento de estilos e lógica de programação. Adicionalmente, é fundamental que os aprendizes adquiram a experiência de trabalhar tanto com sócios que lhes transmita conhecimento quanto com sócios aos quais possa ensinar algo.

7.2.2.3 A obsessão pelo mais experiente

Em sala de aula, existem os aprendizes mais experientes em lógica e os mais experientes em linguagem. Quando os alunos percebem o potencial de aprendizado da técnica de Programação em Duplas, eles procuram se juntar aos mais experientes. A visão de que é possível aprender melhor com sócios experientes é equivocada. Embora exista muito potencial para validá-la, existe também uma grande possibilidade de não ocorrer conflito sócio-cognitivo devido ao grande desequilíbrio de habilidades entre os sócios. Quando não há conflito, o sócio experiente torna-se um mero professor particular. Há de se considerar, à primeira vista, que a presença de um professor particular é de grande valia. Contudo, habilidades em programação não são adquiridas “apenas” pelo simples método de observação, mas sim com muito esforço na prática de resolução de problemas. O conflito força os sócios a se empenharem conjuntamente nas soluções e, conseqüentemente, gera conhecimento empírico.

7.2.2.4 A dificuldade de criar seções de emparelhamento extraclasse

Percebeu-se imensa dificuldade em estender as seções de emparelhamento da sala de aula prática para os trabalhos extraclasse. Contudo, constantemente os aprendizes foram incentivados a produzirem esboços de suas soluções individualmente (em momentos em que estão impossibilitados de emparelhar) e a reunirem-se quando possível no intuito de construir a solução final, fruto de análise e discussão sobre os esboços individuais.

7.2.2.5 As discordâncias

Discordâncias acontecem. Problema maior ocorre quando a discordância é tão acentuada que dificulta o processo de decisão. Nestes casos, o ideal é incentivar os sócios à procura de provas que refutam as próprias idéias. Refutar a idéia do sócio e procurar indícios de que os próprios pensamentos são corretos é muito comum. Contudo, quando se procura refutar o próprio ponto de vista, é possível se colocar na posição do outro. Em muitas situações, ter uma visão apurada sobre ambos os pontos de vista pode ajudar em uma tomada de decisão mais consciente.

7.2.2.6 O desequilíbrio de habilidades

Foram detectadas situações em que o desequilíbrio de habilidades entre os sócios não foi bem aceito pelo membro mais experiente da dupla. Algumas reclamações do tipo “Meu parceiro não sabe nada... assim fica difícil, pois tenho que parar meu raciocínio toda hora para explicá-lo algo...” foram inevitáveis, embora, pouco frequentes. É realmente esperado que nem todos os indivíduos desenvolvam paciência para ensinar e que muitos não apreciam esta habilidade. Contudo, tal ocorrência pode ser uma ótima oportunidade de mudança de comportamento. Afinal, esta é uma habilidade que o profissional de informática precisa desenvolver para se colocar no mercado. Em casos extremos, quando o aprendiz menos experiente não demonstrou nenhum interesse em se aprimorar, procurou-se emparelhá-lo com sócios idênticos. Assim, alunos sem interesse no aprendizado compuseram duplas com alunos também sem interesse. Esta abordagem funcionou na maioria dos casos, enquanto proporcionou que saíssem de sua zona de conforto.

7.2.2.7 A personalidade introvertida e a personalidade egocêntrica

Houve situações atípicas em que alguns aprendizes se disseram céticos sobre os emparelhamentos. Percebeu-se que existem indivíduos com personalidades introvertidas e egocêntricas a ponto de torná-los céticos em relação ao trabalho colaborativo. Programação em Duplas, nestes casos, tende a falhar como recurso pedagógico. Em alguns casos pode, inclusive, se tornar motivo de rejeição à disciplina de Programação. Não foram encontradas soluções para esse problema, embora se procurou incentivá-los à mudança de comportamento demonstrando os benefícios do trabalho colaborativo. Houve apenas um aluno que preferiu veementemente trabalhar de maneira individual – e sua vontade foi respeitada.

7.2.2.8 A dominância/complacência

Enquanto alguns indivíduos apresentaram personalidades dominantes, outros foram considerados demasiadamente complacentes. Os dominantes são aqueles tendenciosos a não aceitar a opinião do sócio e a impor sua vontade como verdade absoluta. Os complacentes são aqueles tendenciosos a aceitar tudo o que lhes é apresentado sem questionamento. Essas duas personalidades podem ser consideradas maléficas ao aprendizado colaborativo. Quando indivíduos com essas personalidades são emparelhados, é possível que o processo de colaboração não alcance seus objetivos. Algumas reuniões específicas com aprendizes que apresentavam essas personalidades foram necessárias. Contudo, concluiu-se que este foi o comportamento maléfico à colaboração mais complexo de se abordar junto aos aprendizes. Estudos minuciosos sobre dominância e complacência durante atividades de Programação em Duplas se fazem necessários.

7.3 ALGUMAS OBSERVAÇÕES PARTICULARES

É importante destacar que os alunos consideraram interessante adquirir conhecimento sob o processo evolutivo de aprendizado proposto pelo MTEA. Embora se percebeu uma grande dificuldade em alcançar a terceira fase de aprendizado em alguns conteúdos (devido ao pouco tempo disponível ou à alta complexidade que este nível impõe), a maioria dos aprendizes relatou satisfação em compreender o estágio de conhecimento em que se encontram em cada momento durante o curso da disciplina.

CAPÍTULO 8

CONCLUSÕES E TRABALHOS FUTUROS

Iniciou-se o presente trabalho com o objetivo de construir um método de ensino-aprendizagem de programação baseado nos fundamentos da metodologia ágil de desenvolvimento de software conhecida como Programação Extrema (*eXtreme Programming*) e de comprovar que o uso de Programação em Duplas é determinante para diminuir a evasão nos períodos iniciais em cursos de Computação.

O método foi construído e nomeado como Aprendizado Extremo de Programação (*eXtreme Programming Learning* – XLP). Foram realizados experimentos em que se aplicou o XLP a uma turma de “Algoritmos e Técnicas de Programação” do 1º semestre de 2008. Os resultados foram processados e comparados a outras 19 turmas (desde o ano 2000) que não trabalharam sob o método XLP. Concluiu-se que a evasão se mostrou inferior na turma experimental quando comparada a 17 destas turmas e que, em 8 delas, a diferença foi considerada estatisticamente significativa.

Contudo, e conforme observado anteriormente, abandonou-se o objetivo de comprovar que o uso de Programação em Duplas é determinante para diminuir a evasão nos períodos iniciais de cursos de Computação. Essa decisão se justifica pelo seguinte questionamento: seria a desmotivação o único motivo ou o mais determinante a promover evasão nas disciplinas de programação? Percebeu-se que seria necessário aplicar questionários aos alunos que se evadiram no intuito de medir o grau de responsabilidade do fator desmotivação neste contexto. A viabilização destes questionários pode ser considerada constrangedora para os desistentes e, provavelmente, impossível de ser realizada, uma vez que a grande maioria deles abandonou também o curso, e não somente a disciplina.

Foram feitos questionamentos sobre a natureza do método, enquanto se parecia muito mais a uma nova estratégia do que a um método de fato. Concluiu-se então que era necessário apoiar em fortes estudos científicos para embasar o método de aprendizado almejado. Neste momento, surgiu a idéia de trabalhar com a taxionomia de objetivos educacionais de Bloom. Enquanto os objetivos sob o domínio cognitivo poderiam ser mapeados a fim de tornar o método mais conciso, os objetivos sob o domínio afetivo poderiam ajudar a explorar os benefícios da motivação durante o ensino de programação. Assim foi feito. Reorganizou-se a taxionomia de objetivos educacionais para o domínio cognitivo em apenas três fases (Método Trifásico de Ensino-Aprendizagem), de forma que as duas primeiras categorias de Bloom (Conhecimento e Compreensão) formaram a primeira fase de aprendizado (Fase 1 – Vejo, Escuto e Compreendo); enquanto as duas categorias intermediárias (Aplicação e Análise) compuseram a segunda fase (Fase 2 – Discuto, Aplico e Reflito); e as duas últimas categorias (Síntese e Avaliação) formaram a terceira fase (Fase 3 – Pratico, Faço e Crio).

Em seguida, foram construídas tabelas mapeando os objetivos a serem alcançados sob os domínios cognitivo e afetivo de acordo com o Método Trifásico de Ensino-Aprendizagem. O conteúdo programático utilizado para o mapeamento corresponde ao conteúdo explanado na disciplina “Algoritmos e Técnicas de Programação” do curso de Sistemas de Informação da Universidade onde foram realizados os experimentos.

O método foi aplicado em outras turmas, no 1º e no 2º semestre de 2009. Neste momento, percebeu-se que aplicá-lo à alunos iniciantes não se mostrou como algo trivial. Mudanças de postura, tanto dos aprendizes, quanto dos professores, foram exigidas constantemente. Por este motivo, e por acreditar que o método pode ajudar a mudar o cenário atual do ensino de programação, iniciou-se a produção de um livro de Algoritmos e Técnicas de Programação totalmente baseado no MTEA. As seguintes observações justificam sua produção:

- a) Diversos estudos sobre problemas no ensino de programação foram realizados, entretanto, foram pouco difundidos no meio acadêmico. Interessados em conhecê-los são obrigados a: realizar pesquisas na Internet ou em periódicos especializados; compilar os resultados, na maioria dos casos, pouco estruturados; e definir estratégias que direcionam como aplicá-los na prática.

- b) Enquanto muitos professores se mostram preocupados em tornar suas aulas enriquecedoras, outros, acomodados pelo ofício, se mantêm inertes quando se deparam com turmas desmotivadas, desinteressadas. É preciso incentivá-los a sair de sua zona de conforto. Isto pode ser feito de posse de um material didático baseado no MTEA.

É importante destacar que os alunos não precisam ser submetidos a todos os níveis do MTEA, embora seja ideal que transpassem as três fases de aprendizado. Turmas demasiadamente fracas podem exigir um tempo maior de maturação, enquanto, forçar aprendizes a alcançar níveis de aprendizado para os quais não se encontram temporariamente aptos pode culminar em um processo de ensino extremamente desmotivador, pois pode trazer à mente questionamentos maléficos ao aprendiz como, por exemplo: “sou incapaz de aprender programação?”; “sou desprovido de inteligência o suficiente para não conseguir atingir tal nível de aprendizado?”; etc.

Fato é, o ambiente de ensino-aprendizagem de programação com o uso do MTEA se mostrou bastante descontraído e mudou a concepção de vários alunos que outrora construíram certa aversão à disciplina. Muitos destes alunos relataram que adquiriram gosto por programação devido ao método.

Esta satisfação pode ser consequência, dentre muitos outros fatores, do processo de aprendizado evolutivo que o MTEA impõe. Os alunos apreciaram conhecer os estágios de aprendizado, principalmente por se tornarem cômicos do “quê” precisam fazer ou conhecer para transpassar cada etapa do conhecimento sobre determinado assunto.

Contudo, e conforme mostrado no capítulo 7, houve uma imensa dificuldade em construir, junto aos alunos, uma cultura para o trabalho colaborativo. Este aspecto demonstra o quanto as pessoas se mostram competitivas e individualistas. Este foi, sem dúvida, o segundo maior problema enfrentado durante a aplicação do MTEA.

Para amenizar este problema, contruiu-se uma cartilha do que podem ser consideradas boas atitudes de profissionais e aprendizes que serão submetidos a técnicas de trabalho colaborativo – embora ela tenha sido contruída especificamente para trabalhos baseados na Programação em Duplas, a cartilha pode ser usada em outros ambientes colaborativos. É

importante destacar que a cartilha foi usada a partir da segunda turma que foi submetida ao MTEA, e mostrou-se útil em vários momentos.

Percebeu-se um comportamento dos aprendizes, muito relatado na literatura, que pode ter sido fundamental para mantê-los focados nos seus objetivos: a pressão das duplas. Curiosamente, e principalmente quando trabalhando com sócios com os quais possuem afinidade, os aprendizes construíram um senso de responsabilidade para com o colega de dupla. Em muitos casos, ficou notório que os sócios demonstraram preocupação com o aprendizado mútuo, com a corretude e com a finalização dos trabalhos durante as seções de laboratório. Alunos dispersos, em muitas situações, foram forçados pelo comportamento de pressão das duplas a se manterem focados nas atividades. Esta ocorrência deve ser considerada positiva, pois, de outra maneira, seria difícil impedir que eles se dispersassem.

Outro comportamento ocorrente é o trabalho conjunto. Observou-se que os alunos, trabalhando conjuntamente, avaliaram várias possibilidades antes de construir a solução para os problemas propostos. Este comportamento é interessante porque faz os alunos transpassarem a segunda fase e, em alguns casos, a terceira fase do MTEA. Adicionalmente, abstrair e analisar soluções específicas pode tornar os aprendizes aptos a construir conhecimentos genéricos que servirão de insumo para a resolução de problemas novos.

Notou-se também que os alunos se apresentaram com maior coragem para participar nos momentos de explanação dos conteúdos (pelo professor) no quadro branco. Em algumas situações, foram vistas duplas tirando dúvidas de outras duplas. Embora em momento algum se procurou incentivar tal comportamento, ele ocorreu frequentemente, e foi muito benéfico.

Um comportamento nomeado como comportamento de depuração em duplas foi percebido. Alguns alunos relataram ter encontrado soluções quando explicavam ao sócio ou ouviam dele explicações de algo relacionado ao problema (uma idéia, uma perspectiva, uma indagação, etc.). Isto ocorreu como *Insights*. Estes *Insights* trouxeram maior senso de realização e satisfação, culminando em motivação.

Como era de se esperar, enfrentou-se dificuldades que precisam ser observadas (quando da implantação do MTEA) e servem de insumos de trabalhos futuros, dentre elas:

- a) Alguns alunos, depois de encontrarem seu sócio de maior afinidade, se tornaram dependentes a ponto de não aceitar experiência com outros sócios. Tentou-se vencer esta dificuldade incentivando a rotatividade, contudo, ainda assim, alguns participantes se mantiveram céticos a este respeito. Nestes casos extremos, decidiu-se respeitar a vontade dos aprendizes – com o objetivo de manter a motivação – embora foi reafirmada em diversas ocasiões a importância da rotatividade das duplas.
- b) Aconteceram casos em que os alunos não se mostraram receptivos à Programação em Duplas. Encontrou-se um aluno introvertido ou egocêntrico a ponto de recusar veementemente trabalhar com um sócio. Este é um comportamento que merece muita atenção, contudo, acredita-se que o mesmo perpassa por questões psicológicas que o torna, provavelmente, um caso a ser estudado por psicólogos.
- c) Situação de dominância/complacência. Foram encontrados alunos dominantes a ponto de não aceitar, até mesmo, a opinião dos professores. Foi interessante colocá-los em dupla com alunos de mesma personalidade – embora não se estudou o comportamento resultante. O maior problema ocorreu com os complacentes. Chegou-se à seguinte indagação: Seria a complacência, em alguns casos, um aspecto da personalidade do indivíduo ou desinteresse ou resistência?

Um fato que preocupa os educadores há tempos, que ficou ainda mais notório nos experimentos, é a observação de que os aprendizes estão adentrando os cursos de Computação com uma bagagem matemática e de leitura e interpretação de textos muito pobre. São poucos os aprendizes que apresentam habilidades de resolução de problemas. É claro que este panorama pode ter sido realçado porque os experimentos foram realizados em uma Universidade da iniciativa privada. Trabalhos futuros podem objetivar a exploração do panorama, sob este aspecto, em Universidades públicas. Isso poderia ajudar a responder questões como as que se seguem: Será este aspecto responsável pelas dificuldades no ensino-aprendizagem de programação? Em que grau? Como e em que instância atacá-lo?

Quanto ao Método Trifásico de Ensino-Aprendizagem, percebeu-se que sua implantação exige comprometimento e despende tempo e esforço. Contudo, é importante destacar as vantagens do seu uso no sentido de tornar o processo de ensino mais controlado e gerenciável, tanto para os alunos quanto para os professores.

É fundamental também a consciência de que as turmas devem ser submetidas ao MTEA de acordo com o contexto no qual estão inseridas. Por exemplo, em turmas cuja disciplina “Programação de Computadores” compõe cursos específicos de Computação, como Sistemas de Informação, Ciência da Computação, Engenharia de Computação, entre outros, deve ser aconselhado que os alunos transpassem, para a maioria do conteúdo programático, as três fases propostas no método. Por outro lado, em turmas em que esta disciplina se insere como disciplina auxiliar e, portanto, compõe cursos diversos de Engenharia, Licenciatura em Computação, entre outros, é aconselhado que o professor sinta-se livre para decidir o nível de explanação dos conhecimentos para aquela classe e, portanto, que fases deseja transpassar com a mesma.

Existe uma incógnita que não pode ser desprezada. Seria a motivação do professor aspecto determinante para uma aplicação bem sucedida do MTEA? É possível que professores não empenhados e não motivados não consigam alcançar resultados satisfatórios ao aplicarem o Método Trifásico de Ensino-Aprendizagem em suas disciplinas. Esta hipótese demonstra a responsabilidade que o educador tem para com o sucesso do método. Contudo, como descrito anteriormente, é esperado que o educador se envolva mais no processo de ensino e abandone sua zona de conforto. Desta maneira, o uso do método pode ser considerado benéfico não somente aos alunos, mas também aos professores.

Finalizando, sugere-se a criação de ferramentas (baseadas no MTEA) de tutoria inteligente de programação. É importante ressaltar que estas ferramentas podem ser construídas a partir do uso de bases de conhecimento de forma que possam ser utilizadas como apoio ao ensino de outras disciplinas de programação, intermediárias ou avançadas.

REFERÊNCIAS

1. ADEMIR DE JESUS, E.; RAABE, A. L. A. **Interpretações da taxionomia de Bloom no contexto da programação introdutória.** XX Simpósio Brasileiro de Informática na Educação (*SBIE 2009*). Florianópolis, Santa Catarina. 17-20 Nov., 2009.
2. AHARONI, D. **Cogito, Ergo Sum! Cognitive Processes of Students Dealing With Data Structures.** Proceedings of the thirty-first SIGCSE technical symposium on Computer science education, 2000.
3. ANDERSON, J. R.; and REISER, B. J. ***The LISP Tutor***. BYTE, April, 1985.
4. ANDERSON, L. W.; KRATHWOHL, D. R; AIRASIAN, P. W.; CRUISHANK, K. A.; MAYER, R. E.; PINTRICH, P. R.; RATHS, J.; and WITTROCK, M. C. **A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives.** New York: Longman, 2001.
5. ARISHOLM, E.; GALLIS, H.; DYBÅ, T.; and SJOBERG, D. I. K. ***Evaluating pair programming with respect to system complexity and programmer expertise.*** IEEE Transactions on software engineering, Vol. 33, Num. 2, February 2007.
6. BECK, L. L.; CHIZHIK, A. W.; and MCELROY, A. C. **Cooperative learning techniques in CS1: design and experimental evaluation.** Proceedings of the 36th SIGCSE technical symposium on Computer science education, February 23-27, St. Louis, Missouri, USA. 2005.
7. BENTLY, J. L.; and KERNINGHAN, B. W. ***A system for algorithm animation.*** Computing Systems, Vol. 4, Num. 1, 1991.

-
8. BLOOM, B.S.; ENGELHART, M.D.; FURST, E.J.; HILL, W.H.; and KRATHWOHL, D.R. **Taxonomy of Educational Objectives, Handbook I: Cognitive Domain**. New York: McKay, 1956 (Tradução Editora Globo, 1972).
 9. BLOOM, B.S.; KRATHWOHL, D.R.; and MASIA, B.B. **Taxonomy of Educational Objectives, Handbook II: Affective Domain**. New York: McKay, 1964 (Tradução Editora Globo, 1973).
 10. BORGES, M. A. F. **Avaliação de uma Metodologia Alternativa para a Aprendizagem de Programação**. VIII Workshop de Educação em Computação, XX Congresso anual da Sociedade Brasileira de Computação, Curitiba, PR. 2000.
 11. BOWER, M. **A Taxonomy of Task Types in Computing**. ITiCSE'08, June 30 – July 2, Madrid, Spain. 2008.
 12. BROWN, M. **Exploring algorithms using Balsa-II**. IEEE Computer. Vol. 21, Num. 5, 1988.
 13. CASTRO, T.; CASTRO JR, A.; e MENEZES, C. **Aprende – um Ambiente Cooperativo de Apoio à Aprendizagem de Programação**. In: Anais do XV Simpósio Brasileiro de Informática na Educação. pp. 71-79. Editora EDUA. Manaus, Brasil. 2004.
 14. CHONG, J.; and HURLBUTT, T. **The Social Dynamics of Pair Programming**. Proceedings of the 29th International Conference on Software Engineering (ICSE'07). Minneapolis, Minnesota. 20-26 May, 2007; pp. 354-363.
 15. CIOLKOWSKI, M.; and SCHLEMMER, M. **Experiences with a Case Study on Pair Programming**. First International Workshop on Empirical Studies in Software Engineering, Finland, 2002.
 16. CLIBURN, D.C. **Experiences with pair programming at a small college**. Journal of Computing Sciences in Colleges, Vol. 19, Issue 1, 2003. pp.20-29.

-
17. COCKBURN, A.; and WILLIAMS, L. **The Costs and Benefits of Pair Programming.** In *Extreme Programming Examined*, Succi, G., Marchesi, M. eds., pp. 223-248, Boston, MA: Addison Wesley, 2001.
 18. CONSTANTINE, L.L. **Constantine on Peopleware.** Yourdon Press Computing Series, ed. E. Yourdon. 1995, Englewood Cliffs, NJ: Yourdon Press.
 19. COPLIEN, J. O. **A Development Process Generative Pattern Language.** In *Pattern Languages of Program Design*, J. O. Coplien and D. C. Schmidt, Ed. Reading Mass: Addison-Wesley, pages 183-237, 1995.
 20. CUBRANIC, D.; STOREY, M. A. D.; and RYALL, J. **A Comparison of Communication Technologies to Support Novice Team Programming.** ICSE'06, May 20-28, Shanghai, China, 2006.
 21. DAVIES, I. **Competence Based Learning.** McGraw Hill, 1973.
 22. DECLUE, T.H., *Pair programming and pair trading: effects on learning an motivation in a CS2 course.* Journal of Computer Sciences in Colleges, Vol. 18, Issue 5, 2003. pp. 49-56.
 23. DEITEL, H. M.; e DEITEL, P. J. **C++: como programar.** 3. ed. Porto Alegre: Bookman, 2001.
 24. DORAN, M. V.; and LANGAN, D. D. **A cognitive-based approach to introductory computer science courses: lesson learned.** Proceedings of the twenty-sixth SIGCSE technical symposium on Computer science education, Nashville, Tennessee, United States, ACM Press, 1995.
 25. FARIA, E. S. J.; VILELA, J. M.; e COELLO, J. M. A. *Um Sistema de Aprendizado Colaborativo de Programação Baseado em Agentes chamado Learn In Group.* XIII Workshop de Educação em Computação (WEI'2005). São Leopoldo, RS, Brasil. 2005.

-
26. FERRANDIN, M.; and STEPHANI, S.L. ***Ferramenta para o ensino de Programação via Internet***. In: SULCOMP - I Congresso Sul Catarinense de Computação, Criciúma, 2005.
27. FESTINGER, L. ***A theory of Cognitive Dissonance***. Evanston, Ill, Row, Peterson, 1957.
28. FORBELLONE, A. L. V.; e EBERSPACHER, H. F. ***Lógica de programação: a construção de algoritmos e estruturas de dados***. 2.ed. São Paulo: Makron Books, 2000.
29. FULLER, U.; JOHNSON, C. G.; AHONIEMI, T.; CUKIERMAN, D.; HERNÁN-LOSADA, I.; JACKOVA, J.; LAHTINEN, E.; LEWIS, T. L.; THOMPSON, D. M.; RIEDESEL, C.; and THOMPSON, E. ***Developing a computer science-specific learning taxonomy***. ACM SIGCSE Bulletin, v.39 n.4, December, 2007.
30. GEHRINGER, E. F.; DEIBEL, K.; HAMER, J.; and WHITTINGTON, K. J. ***Cooperative learning: beyond pair programming and team projects***. Proceedings of the 37th SIGCSE technical symposium on Computer science education. Houston, Texas, USA. ACM SIGCSE Bulletin. Volume 38, Issue 1, March 2006. pp. 458–459.
31. GIRAFFA, L.; MARCZAK, S.; e ALMEIDA, G. ***O ensino de algoritmos e programação mediado por um ambiente na Web***. In: XXIII Congresso Nacional da Sociedade Brasileira de Computação, Campinas, 2003.
32. HAMZE, A. ***A Taxionomia e os objetivos educacionais: A herança educacional de Benjamin Bloom***. [2009]. Canal do educador. Disponível em <<http://www.educador.brasilecola.com/trabalho-docente/a-taxionomia-e-os-objetivos-educacionais.htm>>. Acesso em: 16 de junho de 2009.
33. HANKS, B. F. ***Tool Support for Distributed Pair Programming***. Workshop on Distributed Pair Programming. Extreme Programming and Agile Methods - XP/Agile Universe, 2002.
34. HANKS, B. F. ***Student attitudes toward pair programming***. SIGCSE - Conference on Innovation and Technology in Computer Science Education (ITiCSE '06), 2006. pp. 113-117.

-
35. HANKS, B.; MCDOWELL, C.; DRAPER, D.; and KRNJAJIC, M.. **Program Quality with Pair Programming in CS1**. SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '04), 2004. pp.176-180.
36. HARROW, A. **A taxonomy of the Psychomotor domain: A guide for developing behavioral objectives**. New York: David McKay, 1972.
37. HO, C.; SLATEN, K.; WILLIAMS, L.; and BERENSON, S. **Work in progress-unexpected student outcome from collaborative agile software development practices and paired programming in a software engineering course**. Frontiers in Education (FIE'04), 2004. pp. F2C 15-16.
38. HOWARD, R. A.; CARVER, C. A.; and LANE, W. D. **Felder's learning styles, Bloom's taxonomy, and the Kolb learning cycle: tying it all together in the CS2 course**. *Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education*, Philadelphia, Pennsylvania, United States, ACM Press, 1996.
39. JACOB, P. E. **Changing values in college**. New York, Harper, 1957.
40. JOHNSON, W. L.; and SOLOWAY, E. **PROUST: An automatic debugger for Pascal programs**. BYTE, April, 1985.
41. KATIRA, N.; OSBORNE, J.; and WILLIAMS, L. **Towards increasing the compatibility of student pair programmers**, International Conference on Software Engineering (ICSE '05), 2005. pp. 625-626.
42. KÖCHE, J. C. **Fundamentos de metodologia científica: teoria da ciência e prática da pesquisa**. 18ª ed. Petrópolis: Vozes, 2000.
43. LEJEUNE, N. **Critical Components for Successful Collaborative Learning in CS1**. Journal of Circuits, Systems, and Computers (JCSC'19). 2003.

-
44. LISTER, R.; and LEANEY, J. **First Year Programming: Let All the Flowers Bloom.** Proceedings of 5th Australasian Computer Education Conference (ACE2003), Adelaide. Conferences in Research and Practice in Information Technology, Vol. 20. 2003.
45. MAGER, R.F. **Preparing Objectives for Programmed Instruction.** San Francisco, California: Fearon, 1962.
46. MATZKO, S.; and DAVIS, T. **Pair design in undergraduate labs.** Journal of Computing Sciences in Colleges, Vol. 22, Issue 2, 2006. pp.123–130.
47. MCDOWELL, C.; HANKS, B.; and WERNER, L. **Experimenting with pair programming in the classroom.** SIGCSE Conference on Innovation and Technology in Computer Science Education, 2003. pp.60-64.
48. MCDOWELL, C.; WERNER, L.; BULLOCK, H. E.; and FERNALD, J. **The impact of pair programming on student performance, perception and persistence.** Proceedings of the 25th International Conference on Software Engineering, May 03-10, Portland, Oregon. 2003.
49. MCDOWELL, C.; WERNER, L.; BULLOCK, H. E.; and FERNALD, J. **Pair programming improves student retention, confidence, and program quality.** Communications of the ACM, Vol. 49, Num. 8, 2006. pp.90-95.
50. MENDES, A. J.; and MENDES, T. **VIP - A Tool to Visualize programming exemples.** In Proc. Education and Application of Computer Technology. pp 131-140. Malta. 1988.
51. MENDES, E.; AL-FAKHRI, L.; and LUXTON-REILLY, A. **A replicated experiment of Pair Programming in a 2nd-year software development and design computer science course.** ACM SIGCSE Bulletin, v.38 n.3, September 2006.
52. MENDES, E. B. **Tendências futuras para planejamento curricular construtivista no ensino médio.** 2002. 177 f. Tese (Doutorado em Engenharia de Produção) – Centro Tecnológico – CTC, Departamento de Engenharia de Produção e Sistemas, Universidade Federal de Santa Catarina, Florianópolis, 2002.

-
53. MUJEEB-U-REHMAN, M.; YANG, X.; DONG, J.; and ABDUL GHAFOOR, M. **Heterogeneous and homogenous pairs in pair programming: an empirical analysis.** Conference on Electrical and Computer Engineering, 2005. Canadian. 1-4 May 2005; pp. 1116-1119.
54. NAGAPPAN, N.; WILLIAMS, L.; FERZLI, M.; WIEBE, E.; YANG, K.; MILLER, C.; and BALIK, S. **Improving the CS1 Experience with Pair Programming.** ACM Technical Symposium on Computer Science Education, SIGCSE, 2003. pp.359-362.
55. NAWROCKI, J.; and WOJCIECHOWSKI, A. **Experimental Evaluation of Pair Programming.** Presented at European Software Control and Metrics (ESCOM 2001), London, England, 2001.
56. NOSEK, J.T. **The Case for Collaborative Programming.** In Communications of the ACM. 1998. pp. 105-108.
57. PADBERG, F.; and MULLER, M. M. **Analyzing the Cost and Benefit of Pair Programming.** Proceedings of the 9th International Symposium on Software Metrics, p.166, September 03-05, 2003.
58. PRATA, C. L.; NASCIMENTO, A. C. A. A. **Objetos de aprendizagem: uma proposta de recurso pedagógico.** Ministério da Educação. Secretaria de Educação a Distância. Brasil. 2007. 154p. ISBN: 978-85-296-0093-2.
59. PRESTON, D. **Pair programming as a model of collaborative learning: a review of the research.** *Journal of Computing Sciences in Colleges*, Vol. 20, Num. 4, 2005. pp.39-45.
60. PRESTON, D. **Using collaborative learning research to enhance pair programming pedagogy.** ACM SIGITE Newsletter, Vol. 3, Issue 1, January 2006. pp.16–21.

-
61. PROULX, V.K. ***Programming Patterns and Design Patterns in the Introductory Computer Science Course***. Proceedins of the 31st SIGCSE, pp. 7-12, Auxtin, TX, USA, march, 2000.
62. SALLEH, N.; MENDES, E.; and GRUNDY, J. ***Empirical Studies of Pair Programming for CS/SE Teaching in Higher Education: A Systematic Literature Review***. IEEE Transactions on Software Engineering, June, 2010.
63. SANTOS, R. P.; e COSTA, H. A. X. ***Análise de Metodologias e Ambientes de Ensino para Algoritmos, Estruturas de Dados e Programação aos iniciantes em Computação e Informática***. p.41-50. INFOCOMP – Journal of Computer Science. Vol. 5, Num. 1. 2006.
64. SCHEERER, M. ***Cognitive theory***. In: Handbook of social psychology. Cambridge, Mass.; Addison-Wesley, vol I, cap 3. 1954.
65. SELKER, T. ***COACH: A teaching Agent that learns***. Communications of the ACM. Vol 37, No. 7. 1994.
66. SHERRELL, L. B.; and ROBERTSON, J. J. ***Pair programming and agile software development: experiences in a college setting***. Journal of Computing Sciences in Colleges, Vol. 22, Issue 2, 2006. pp.145-153.
67. SIMPSON, B. J. ***The classification of educational objectives: Psychomotor domain***. Illinois Journal of Home Economics, vol. 10, num. 4, 1966. pp. 110-144.
68. SONG, J. S.; HAHN, S. H.; TAK, K. Y.; and KIM, J. H. ***An intelligent tutoring system for introductory C language course***. Computers & Education. Vol 28, No. 2. 1997.
69. STASKO, J.; and MCCRICKARD, D. ***Real clock time animation support for developing software visualizations***. Australian Computer Journal, Vol. 27, Num. 3, 1995.

-
70. TEDESCO, P. C. A. R. **Mediating Meta-Cognitive Conflicts in Group Planning Situations**. 2001. 280 f. Thesis (Doctoral of Philosophy) – Computer Based Learning Unit and School of Computing, University of Leeds, Leeds, united Kingdom, 2001.
71. THOMPSON, E.; HUNT, L.; and KINSHUK. **Exploring Learner Conceptions of Programming**. Australian Computer Science Communications, 52. 2006. pp. 205 - 211.
72. THOMPSON, E.; LUXTON-REILLY, A.; WHALLEY, J.; HU, M.; and ROBBINS, P. **Bloom's taxonomy for CS assessment**. Proceedings of the Tenth Australasian Computing Education Conference (ACE2008), Wollongong, Australia, January, 2008.
73. TOBAR, C. M.; ROSA, J. L. G.; COELLO, J. M. A.; PANNAIN, R. ***Uma Arquitetura de Ambiente Colaborativo para o Aprendizado de Programação***. XII Simpósio Brasileiro de Informática na Educação, Vitória - ES. 2001.
74. TOMEK, I. ***Josef the Robot***. Computers & Education, Vol. 6, Num. 3, 1982.
75. TOMEK, I.; MULDER, T.; and KHAN, S. ***PMS - A program to make learning Pascal easier***. Computers and Education. Vol 9, No. 4. 1985.
76. VANHANEN, L. K. **Experiences of Using Pair Programming in an Agile Project**. In 40th Annual Hawaii International Conference on System Sciences (HICSS'07), 2007.
77. WERNER, L.; DENNER, J.; and BEAN, S. **Pair Programming Strategies for Middle School Girls**. IASTED International Conference on Computers and Advanced Technology in Education (CATE '04), 2004. pp. 161-166.
78. WERNER, L.; HANKS, B.; and MCDOWELL, C. **Pair programming helps female computer science students persist**. ACM Journal of Educational Resources in Computing, 4(1), Mar. 2004. Published April 2005.

-
79. WILLIAMS, L. **Integrating Pair Programming into a Software Development Process.** CSEET'01, p. 27, 14th Conference on Software Engineering Education and Training, 2001.
80. WILLIAMS, L. **Debunking the Nerd Stereotype with Pair Programming.** (Broadening Participation in Computing Series), IEEE Computer, Vol. 39, Issue 5, 2006. pp.83-85.
81. WILLIAMS, L.; and KESSLER, R.R. **All I Really Need to Know about Pair Programming I Learned In Kindergarten.** Communications of the ACM, May, 2000.
82. WILLIAMS, L.; and KESSLER, R.R. **Experimenting with Industry's Pair Programming Model in the Computer Science Classroom.** Computer Science Education, Vol. 11, Issue 1. 2001. pp. 7-20.
83. WILLIAMS, L.; KESSLER, R. R.; CUNNINGHAM, W.; and JEFFRIES, R. **Strengthening the Case for Pair Programming.** IEEE Software, vol. 17, no. 4, pp. 19-25, Jul/Aug, 2000.
84. WILLIAMS, L.; LAYMAN, L.; OSBORNE, J.; and KATIRA, N. **Examining the Compatibility of Student Pair Programmers.** (AGILE'06), 2006. pp. 411-420.
85. WILLIAMS, L.; LAYMAN, L.; SLATEN, K. M.; BERENSON, S. B.; and SEAMAN, C. **On the Impact of a Collaborative Pedagogy on African American Millennial Students in Software Engineering.** 29th International Conference on Software Engineering, 2007. ICSE'07. 20-26 May, 2007. pp. 677-687.
86. WILLIAMS, L.; MCDOWELL, C.; FERNALD, J.; WERNER, L.; and NAGAPPAN, N. **Building Pair Programming Knowledge Through a Family of Experiments.** IEEE International Symposium on Empirical Software Engineering (ISESE), 2003. pp.143-152.
87. WILLIAMS, L.; WIEBE, E.; YANG, K.; FERZLI, M.; and MILLER, C. **In Support of Pair Programming in the Introductory Computer Science Course.** Computer Science Education, Vol. 12, Issue 3, September 2002. pp.197-212.

-
88. WHITTINGTON, K. J. ***Infusing active learning into introductory programming courses***. Journal of Computing Sciences in Colleges, Vol. 19, Num. 5, 2004. pp.249-259.
89. YUKSEL, A. **Pair programming: matching pairs in a sequence of learning sessions**. 8th Human Centred Technology Postgraduate Workshop (Advancing the Potential for Communication, Learning and Interaction), June 28th - 29th 2005.
90. ZIN, A. M.; IDRIS, S.; and SUBRAMANIAM, N. K. ***Improving Learning of Programming Through E-Learning by Using Asynchronous Virtual Pair Programming***. The Turkish Online Journal of Distance Education, Vol. 7, Issue 3, 2006. pp.162-173.
91. ZUALKERNAN, I. A. **Using Soloman-Felder Learning Style Index to Evaluate Pedagogical Resources for Introductory Programming Classes**. Proceedings of the 29th International Conference on Software Engineering (ICSE'07). Minneapolis, Minnesota. 20-26 May, 2007. pp. 723-726.

APÊNDICE A

EXERCÍCIOS PROPOSTOS PARA APLICAÇÃO DURANTE AS FASES DO MTEA

Os exercícios disponíveis nas seguintes seções são exemplos de atividades que podem ser usadas como material auxiliar durante o ensino-aprendizagem de programação ou como meio de avaliar os aprendizes durante ou após transpassar as fases de aprendizagem que formam o MTEA. Os exercícios foram classificados, primeiramente, de acordo com os tópicos do conteúdo programático da disciplina “Programação de Programadores” e, em seguida, por fase de aquisição de conhecimento – de acordo com a metodologia trifásica de aprendizado.

A.1. COMPREENDER OS SISTEMAS DE NUMERAÇÃO E AS TRANSFORMAÇÕES DE BASE

Exercícios da fase “Vejo, Escuto e Compreendo”

- 1) O que é um sistema de numeração?
- 2) Por que nosso sistema de numeração é decimal?
- 3) Realize uma busca na Internet em busca de curiosidades sobre os seguintes sistemas de numeração: decimal, binário, octal e hexadecimal. Elabore um relatório com várias curiosidades encontradas.

- 4) Identifique que (provável ou prováveis) sistema(s) de numeração cada número abaixo pode representar:
- | | | |
|-----------|----------|-----------|
| a. 123 | e. 1AF10 | i. 1991 |
| b. 8AF | f. 17 | j. ABCDEF |
| c. 101110 | g. 17123 | k. 117271 |
| d. 10991 | h. 11FF1 | l. 991190 |
- 5) Transforme os números a seguir (representados na base 10) em números binários:
- | | | |
|-------------------|-------------------|--------------------|
| a. $123_{(10)}$ | e. $2230_{(10)}$ | i. $1991_{(10)}$ |
| b. $813_{(10)}$ | f. $17_{(10)}$ | j. $348123_{(10)}$ |
| c. $2_{(10)}$ | g. $17123_{(10)}$ | k. $117271_{(10)}$ |
| d. $10991_{(10)}$ | h. $9991_{(10)}$ | l. $991190_{(10)}$ |
- 6) Qual é o sistema de numeração dos computadores e por que não é decimal?
- 7) Transforme os números a seguir em números na base hexadecimal:
- | | | |
|------------------|--------------------|--------------------|
| a. $4567_{(10)}$ | e. $12_{(10)}$ | i. $344556_{(10)}$ |
| b. $1010_{(10)}$ | f. $1_{(10)}$ | j. $123_{(10)}$ |
| c. $2981_{(10)}$ | g. $17_{(10)}$ | k. $1111_{(10)}$ |
| d. $16_{(10)}$ | h. $239991_{(10)}$ | l. $121212_{(10)}$ |
- 8) Transforme os números a seguir em números na base octal:
- | | | |
|-------------------|--------------------|--------------------|
| a. $22222_{(10)}$ | e. $1200_{(10)}$ | i. $111111_{(10)}$ |
| b. $6_{(10)}$ | f. $100090_{(10)}$ | j. $8888_{(10)}$ |
| c. $8_{(10)}$ | g. $34_{(10)}$ | k. $77332_{(10)}$ |
| d. $1919_{(10)}$ | h. $2340_{(10)}$ | l. $202012_{(10)}$ |
- 9) Transforme os números a seguir em números na base pré-determinada de acordo com a direção da seta:
- | | | |
|---------------------------------|----------------------------------|--------------------------------|
| a. $22_{(10)} \rightarrow (16)$ | e. $1200_{(8)} \rightarrow (2)$ | i. $11_{(2)} \rightarrow (8)$ |
| b. $6_{(8)} \rightarrow (16)$ | f. $190_{(16)} \rightarrow (2)$ | j. $77_{(8)} \rightarrow (10)$ |
| c. $8_{(16)} \rightarrow (8)$ | g. $110_{(2)} \rightarrow (16)$ | k. $77_{(10)} \rightarrow (2)$ |
| d. $1010_{(2)} \rightarrow (8)$ | h. $141_{(16)} \rightarrow (10)$ | l. $77_{(16)} \rightarrow (2)$ |
- 10) Transforme os números a seguir em números na base pré-determinada de acordo com a direção da seta:
- | | |
|-----------------------------------|---------------------------------------|
| a. $22102_{(10)} \rightarrow (2)$ | e. $177200_{(8)} \rightarrow (2)$ |
| b. $600_{(8)} \rightarrow (10)$ | f. $1900_{(8)} \rightarrow (16)$ |
| c. $800_{(16)} \rightarrow (8)$ | g. $11099198_{(10)} \rightarrow (16)$ |
| d. $90F20_{(16)} \rightarrow (2)$ | h. $141AA_{(16)} \rightarrow (10)$ |

Exercícios da fase “Discuto, Aplico e Reflito”

- 1) Por que números representados por bases menores são formados por uma quantidade maior de algarismos que bases maiores? (Por exemplo, o número 10 na base decimal é formado por dois algarismos, enquanto o correspondente na base binária (1010) é formado por 4 algarismos)
- 2) Por que motivo os números na base hexadecimal devem ser representados por mais seis números em formato de letras (A, B, C, D, E, F)?
- 3) Explique por que 1 byte corresponde a 8 bits ao invés de 10.
- 4) Se 1 km corresponde a 1000 metros, por que 1 kbyte corresponde a 1024 bytes ao invés de 1000?
- 5) Quantos números podem ser representados por 1 byte, por 2 bytes e por 4 bytes? Justifique cada resposta.
- 6) Descreva os 16 primeiros números (em base binária) que um byte pode representar.
- 7) Transforme os números a seguir em números na base pré-determinada de acordo com a direção da seta:
 - a. $22102_{(10)} \rightarrow (20)$
 - b. $600_{(8)} \rightarrow (4)$
 - c. $800_{(16)} \rightarrow (12)$
 - d. $90F20_{(18)} \rightarrow (6)$
 - e. $77200_{(10)} \rightarrow (7)$
 - f. $1900_{(18)} \rightarrow (16)$
 - g. $11099198_{(10)} \rightarrow (25)$
 - h. $141AA_{(14)} \rightarrow (10)$
- 8) Quantos números podem ser representados por:
 - a. 5 algarismos na base 10?
 - b. 8 algarismos na base 16?
 - c. 4 algarismos na base 60?
 - d. 7 algarismos na base 2?
 - e. 3 algarismos na base 20?
 - f. 6 algarismos na base 8?
 - g. 8 algarismos na base 6?
 - h. 2 algarismos na base 17?
- 9) Pesquise e descreva a fórmula generalizada de representação de um número N em uma base posicional qualquer.
- 10) Qual é o maior número par, da base 10, que você pode escrever na base 2 usando apenas 5 algarismos?

Exercícios da fase “Prático, Faço e Crio”
--

- 1) Como realizar cálculos aritméticos entre números representados em bases diferentes?
- 2) Faça uma pesquisa e elabore um resumo sobre o sistema sexagesimal babilônico.
- 3) Realize as operações aritméticas abaixo:

a. $110011_{(2)} + 110011_{(2)}$	f. $100011_{(2)} + 11_{(2)}$
b. $1111_{(2)} + 10011_{(2)}$	g. $110010_{(2)} + 100010_{(2)}$
c. $111_{(2)} + 1100111_{(2)}$	h. $110010_{(2)} - 100010_{(2)}$
d. $11110011_{(2)} - 10011_{(2)}$	i. $11111011_{(2)} + 101_{(2)}$
e. $100011_{(2)} - 11_{(2)}$	j. $100000_{(2)} - 1011_{(2)}$
- 4) Realize as operações aritméticas abaixo:

a. $110011_{(2)} * 110011_{(2)}$	e. $100011_{(2)} / 11_{(2)}$
b. $1111_{(2)} * 10011_{(2)}$	f. $110011_{(2)} * 100010_{(2)}$
c. $111_{(2)} * 1100111_{(2)}$	g. $11111011_{(2)} / 101_{(2)}$
d. $110011_{(2)} / 11011_{(2)}$	h. $100000_{(2)} / 1011_{(2)}$
- 5) Realize as operações aritméticas abaixo:

a. $ABA_{(16)} + B_{(16)}$	e. $1DA_{(16)} + A_{(16)}$
b. $AFE_{(16)} - A_{(16)}$	f. $CDE_{(16)} + 2_{(16)}$
c. $12BA_{(16)} - 10_{(16)}$	g. $BCD1_{(16)} + 3_{(16)}$
d. $101F_{(16)} - 5_{(16)}$	h. $A221_{(16)} - B1_{(16)}$
- 6) Realize as operações aritméticas abaixo:

a. $AAA_{(16)} * B_{(16)}$	e. $11A_{(16)} * A_{(16)}$
b. $A1E_{(16)} / A_{(16)}$	f. $1CDE_{(16)} * 2_{(16)}$
c. $11EA_{(16)} / 10_{(16)}$	g. $BCD_{(16)} * 3_{(16)}$
d. $1010_{(16)} / 5_{(16)}$	h. $A22_{(16)} / B1_{(16)}$
- 7) Realize as operações aritméticas abaixo:

a. $747_{(8)} + 5_{(8)}$	e. $117_{(8)} + 67_{(8)}$
b. $714_{(8)} - 2_{(8)}$	f. $1214_{(8)} + 266_{(8)}$
c. $1447_{(8)} - 210_{(8)}$	g. $31_{(8)} + 32_{(8)}$
d. $1710_{(8)} - 325_{(8)}$	h. $722_{(8)} - 310_{(8)}$
- 8) Realize as operações aritméticas abaixo:

a. $777_{(8)} * 3_{(8)}$	e. $117_{(8)} * 7_{(8)}$
b. $714_{(8)} / 7_{(8)}$	f. $1214_{(8)} * 2_{(8)}$
c. $1147_{(8)} / 10_{(8)}$	g. $321_{(8)} * 3_{(8)}$
d. $1010_{(8)} / 5_{(8)}$	h. $722_{(8)} / 31_{(8)}$

9) Realize as operações aritméticas abaixo e dê o resultado em ambas as bases:

a. $747_{(8)} * 3_{(16)}$

e. $317_{(8)} * 7_{(10)}$

b. $714_{(8)} / 7_{(10)}$

f. $1115_{(8)} * 2_{(16)}$

c. $1247_{(8)} / 10_{(2)}$

g. $371_{(8)} * 3_{(4)}$

d. $5010_{(8)} / 3_{(4)}$

h. $7132_{(8)} / 1001_{(2)}$

10) Realize as operações aritméticas abaixo e dê o resultado na base decimal:

a. $333_{(8)} * 3_{(8)}$

e. $113_{(8)} * 30_{(4)}$

b. $314_{(8)} / 3_{(16)}$

f. $1214_{(8)} * 20_{(16)}$

c. $1143_{(8)} / 10_{(2)}$

g. $321_{(16)} * 3_{(4)}$

d. $1017_{(8)} / 5_{(8)}$

h. $322_{(8)} / 31_{(4)}$

A.2. INTRODUIZIR ALGORITMOS

Exercícios da fase “Vejo, Escuto e Compreendo”

- 1) Faça uma busca na Internet e elabore um resumo sobre o conceito de algoritmo (O que é; história; como construir algoritmos; por que construí-los; entre outras informações).
- 2) Elabore um algoritmo para produzir um bolo de cenoura.
- 3) Elabore um algoritmo para produzir uma lasanha ao molho sugo.
- 4) Elabore um algoritmo para estudar para uma prova de física.
- 5) Elabore um algoritmo para estudar para uma prova de química.
- 6) Elabore um algoritmo para produzir um veículo.
- 7) Elabore um algoritmo para comprar uma passagem no *site* de uma empresa de transporte aéreo (Por exemplo: “Gol”, “TAM”, “Trip”).
- 8) Elabore um algoritmo para se cadastrar em um *site* de comércio eletrônico (Por exemplo: “Americanas.com”, “e-facil.com”, “comprafacil.com”).
- 9) Elabore um algoritmo (em forma textual descritiva) para calcular a soma e a média das idades dos alunos da sua turma.
- 10) Elabore um algoritmo (em forma textual descritiva) para calcular as raízes de uma equação do 2º grau.

Exercícios da fase “Discuto, Aplico e Reflito”

- 1) Que algoritmo é mais fácil de escrever? Algoritmo para produzir uma lasanha ou um veículo? Justifique.
- 2) Por que muitas soluções algorítmicas dos demais colegas de turma não são idênticas à sua e mesmo assim são consideradas corretas?
- 3) Descreva alguns motivos pelos quais construir algoritmos corretos pode não ser trivial.
- 4) Algoritmos devem ser escritos em um nível de detalhamento razoável para que sejam facilmente interpretados e executados. Quando este nível de detalhamento se torna mais prejudicial do que benéfico ao se produzir algoritmos que serão executados por outras pessoas?
- 5) Elabore um algoritmo para produzir um pudim de leite. Em seguida, elabore um algoritmo para produzir uma broa de fubá. Descreva as estruturas que foram usadas e se repetiram na solução dos dois algoritmos (ingredientes e/ou passos). Por que estas estruturas se repetiram? Adicione linhas de comentários onde julgar necessário.
- 6) Construa um enunciado para o algoritmo abaixo, condizente com o que ele se propõe a fazer. Adicione linhas de comentários ao código abaixo onde julgar necessário.

Algoritmo “XXXX”

Início

Procure e pegue 2 maçãs na geladeira
Procure e pegue 3 bananas na fruteira
Procure e pegue 1 mamão na fruteira
Procure e pegue 2 pêras na geladeira
Procure e pegue 6 laranjas na fruteira
Descasque todas as frutas
Pique as maçãs, as pêras e o mamão em uma tigela retirando as sementes
Pique a banana na mesma tigela
Pique as laranjas e esprema o sumo na tigela
Coloque uma colher de sopa de açúcar
Misture com a colher
Sirva à gosto

FimAlgoritmo

- 7) Construa um enunciado para o algoritmo abaixo, condizente com o que ele se propõe a fazer. Adicione linhas de comentários ao código onde julgar necessário.

Algoritmo “YYYY”

Início

Procure e pegue 2 maçãs e duas pêras na geladeira
Procure e pegue 3 bananas, 1 mamão e 6 laranjas na fruteira
Descasque todas as frutas
Pique as maçãs, as pêras e o mamão em uma tigela retirando as sementes
Pique a banana na mesma tigela
Pique as laranjas na mesma tigela
Coloque todo o conteúdo da tigela no liquidificador
Coloque uma colher de sopa de açúcar
Bata a mistura no liquidificador por 3 minutos
Sirva à gosto

FimAlgoritmo

- 8) Identifique os componentes em comum encontrados nos algoritmos da questão 6 e 7. Como se pode ver, a quantidade de componentes em comum entre eles é alta. Que informações é possível concluir a esse respeito? (Generalize sua resposta).
- 9) Analise o código do algoritmo abaixo. Verifique que ele se propõe a fazer uma omelete. Contudo, algumas instruções necessárias à produção de uma omelete foram suprimidas. Corrija o algoritmo incluindo instruções e comentários onde considerar necessário. É importante destacar que o queijo curado se encontra em cima da geladeira e que o ralador se encontra embaixo da pia.

Algoritmo “Omelete”

Início

Procure e pegue 2 ovos na porta da geladeira
Bata os ovos
Coloque queijo ralado
Coloque o óleo na frigideira
Coloque os ovos batidos com o queijo ralado na frigideira (já temperado)
Mexa com uma espátula até formar uma massa mais consistente na frigideira
Virar a omelete

FimAlgoritmo

- 10) Analise o código do algoritmo abaixo. Perceba que ele se propõe a ajudá-lo a dirigir um veículo por 100 metros em sentido à frente. Contudo, o algoritmo está incorreto e, portanto, dirigindo desta maneira, dificilmente você chegará a algum lugar. Descreva as incorreções sintáticas e semânticas do algoritmo. Justifique as correções que considerou como correções semânticas. Reescreva o algoritmo da maneira correta. Adicione linhas de comentário onde julgar importante.

Algoritmo “Dirigindo um automóvel”

Início

- Entrar em um carro
- Abra uma lata de cerveja
- Colocar o chaveiro na ignição
- Apertar o pé no acelerador para ligar o veículo
- Colocar também a chave na ignição
- Virar a chave até o atingir o primeiro estágio da ignição
- Verificar se a marcha está em ponto morto
- Colocar a marcha em ponto morto
- Virar a chave do veículo para o veículo desligar
- Pisar no freio e engatar a 1ª marcha
- Seguir em direção à ré
- Pisar na embreagem com o objetivo de parar o veículo
- Abrir a janela do veículo
- Sair do veículo

FimAlgoritmo

Exercícios da fase “Pratico, Faço e Crio”

- 1) A seguir podem ser vistos dois algoritmos para solução de um único problema (produzir um bolo de abacaxi). Identifique que algoritmo é mais eficiente (em relação ao tempo de produção). Justifique sua resposta.

Algoritmo “**Bolo de Abacaxi - 1**”

Início

- Pegue um saco de farinha de trigo e separe 4 copos americanos em uma tigela
- Pegue 2 ovos na geladeira e quebre-os na tigela
- Bata o conteúdo da tigela no liquidificador
- Adicione fermento, uma pitada de sal e uma colher de margarina
- Adicione uma colher de sopa de açúcar e suco de polpa de abacaxi
- Bata tudo mais uma vez e Unte a forma
- Despeje o conteúdo do liquidificador na forma
- Ligue o forno e leve a forma ao forno por 25 minutos.
- Retire o bolo e deixe-o esfriando por 10 minutos e Coma o bolo à vontade

FimAlgoritmo

Algoritmo “Bolo de Abacaxi - 2”

Início

Pegue um massa pronta de bolo e a despeje em uma tigela
Pegue 2 ovos na geladeira e quebre-os na tigela
Bata o conteúdo da tigela no liquidificador
Adicione uma colher de margarina
Bata novamente
Adicione suco de polpa de abacaxi e bata uma última vez
Unte a forma
Despeje o conteúdo do liquidificador na forma
Ligue o forno
Leve a forma ao forno por 25 minutos.
Retire o bolo e deixe-o esfriando por 10 minutos
Coma o bolo à vontade

FimAlgoritmo

- 2) Construa um algoritmo ainda mais eficiente que os vistos no exercício “1” (para produzir um bolo de abacaxi). Justifique sua solução.
- 3) Construa um algoritmo para produzir uma folha de pagamento de funcionários de uma empresa. Se possível, procure informações detalhadas com um funcionário de um departamento pessoal.
- 4) Escreva um algoritmo para construir uma casa. Depois de pronto, detalhe cada instrução do algoritmo que é formada por uma série de outras instruções (Formando a idéia de macro e mini algoritmos).
- 5) Avalie a utilidade de cada linha de comentário encontrada no algoritmo abaixo e construa comentários mais significativos onde considerar necessário.

Algoritmo “Comprando Doces”

Início

Entrar no supermercado
// Verificar quanto você tem de dinheiro para fazer sua compra é interessante
// Você não trouxe cartão de crédito
Locomover-se até a seção de doces
Procurar prateleira com chocolates
// Chocolates podem ser brancos ou comuns
Escolher chocolates
Somar o preço dos chocolates
// Somar é uma operação aritmética
Pegar chocolates e colocar na sacola de compras
// Uma sacola de compras é como um carrinho de compras (sem rodas)
Se dirigir até o caixa e aguardar na fila
Passar os chocolates no caixa
Pagar os chocolates
// Pagar os chocolates brancos e os comuns
Sair do supermercado

FimAlgoritmo

- 6) Construa um algoritmo (mais eficiente possível) para produzir um suco de mangas. Adicione comentários significativos no algoritmo que justifiquem o fato dele ser considerado eficiente.
- 7) Construa um algoritmo para agendar e comparecer em uma consulta de um profissional dentista. Adicione comentários significativos no algoritmo.
- 8) Construa um algoritmo para descrever as etapas de construção de um algoritmo. Adicione comentários significativos ao algoritmo.
- 9) Construa um algoritmo para abastecer um veículo em um posto de combustível. Adicione comentários onde considerar necessário.
- 10) Peça aos colegas de turma (pelo menos três) uma cópia dos algoritmos de cada um (para os exercícios 6, 7, 8 e 9 desta lista) e avalie (entre o seu algoritmo e os de seus colegas) que algoritmo é mais eficiente dentre todos (para cada questão). Analise (também para cada exercício isoladamente) a significância dos comentários encontrados nas soluções dos colegas.

A.3. CONSTRUIR EXPRESSÕES E ENTENDER A PRIORIDADE DE OPERADORES

Exercícios da fase “Vejo, Escuto e Compreendo”

- 1) Descreva o que são e quais são os operadores: (a) lógicos; (b) aritméticos; (c) relacionais; (d) literais.
- 2) Execute (resolva usando “R” como resposta para cada operação) as expressões abaixo de acordo com a ordem de prioridade dos operadores aritméticos.

a. $3 * 4 - 3$	e. $12 \bmod 3 \operatorname{div} 4 ** 5$
b. $39 * 2 * A - C / 2$	f. $13 + 6 * 5 - (3 + 4)$
c. $21 ** 4 * 5 \bmod 3$	g. $I * 5 - 4 * (5 / 3)$
d. $A - F * 8 \operatorname{div} 4 ** 2$	h. $A ** B ** C \bmod D$
- 3) Execute (resolva usando “R” como resposta para cada operação) as expressões abaixo de acordo com a ordem de prioridade dos operadores aritméticos.

a. $21 ** A - 21$	c. $21 \operatorname{div} A ** 5 + 21$
b. $219 ** 2 ** A - C - 2$	d. $A - B ** 8 \operatorname{div} A \operatorname{div} 2$

- e. $12 + 21 \text{ div } A \text{ div } 5$
 - f. $121 + 6 ** 5 - (21 + A)$
 - g. $I ** 5 - A ** (5 - 21)$
 - h. $A \text{ div } B \text{ div } C + D$
- 4) Execute (resolva usando “R” como resposta para cada operação) as expressões abaixo de acordo com a ordem de prioridade dos operadores relacionais e aritméticos.
- a. $12 > 1 - 12$
 - b. $129 > 2 * G - 13 ** C$
 - c. $21 \leq 1 + 12 ** 2$
 - d. $G = F ** 2 ** 4$
 - e. $12 ** 3 <> 12 * 4 - 7$
 - f. $4 / 23 * 42 \leq 112 / 5 + 8$
- 5) Execute (resolva usando “R” como resposta para cada operação) as expressões abaixo de acordo com a ordem de prioridade dos operadores lógicos.
- a. $V \text{ e } V \text{ ou } F$
 - b. $\neg V \text{ ou } V \text{ e } \neg F$
 - c. $\neg(V \text{ e } F) \text{ ou } F$
 - d. $\neg V \text{ e } \neg F \text{ ou } \neg(V \text{ ou } F)$
 - e. $(V \text{ e } \neg F \text{ ou } \neg V) \text{ e } (F \text{ ou } \neg V)$
 - f. $\neg V \text{ e } V \text{ ou } \neg F \text{ e } (V \text{ ou } F)$
- 6) Execute (resolva mostrando os resultados) as expressões abaixo de acordo com a ordem de prioridade dos operadores literais.
- a. “joão” + “Maria”
 - b. “João da Silva” - “Silva” + “Matos”
 - c. “Maria do Carmo” + “Silva” - “do Carmo”
 - d. “Regina” - “gina” + “nata”
- 7) Execute (resolva usando “R” como resposta para cada operação) as expressões abaixo de acordo com a ordem de prioridade dos operadores aritméticos, relacionais e lógicos.
- a. $3 * 4 > 12 - 3 + 5 ** 2 \text{ e } V$
 - b. $2 ** 2 = 3 \text{ e } \neg(V \text{ ou } F)$
 - c. $A - 23 <> 23 \text{ ou } \neg F$
 - d. $A + 3 > 4 \text{ ou } (3 <> 7) \text{ e } \neg(V \text{ ou } F) \text{ ou } (Z ** 2 > 4)$
- 8) Execute (resolva mostrando os resultados) as expressões abaixo de acordo com a ordem de prioridade dos operadores aritméticos, relacionais e lógicos. Os valores de A e Z abaixo são: A=7; Z=78.
- a. $A * 4 \leq 12 - A \text{ mod } 5 - 2 \text{ e } V \text{ ou } (2 - 2 = A \text{ e } \neg(V \text{ ou } F))$
 - b. $2 - 2 = A \text{ e } \neg(V \text{ ou } F) \text{ e } (A - 2 * A \leq 2 + A \text{ ou } \neg F)$
 - c. $A - 2 * A \leq 2 + A \text{ ou } \neg F$
 - d. $A \text{ mod } A \leq 4 \text{ ou } (A \leq 7) \text{ e } \neg(V \text{ ou } F) \text{ ou } (Z - 2 \leq 4)$

- 9) Execute (resolva mostrando os resultados) as expressões abaixo de acordo com a ordem de prioridade dos operadores aritméticos, relacionais e lógicos. Os valores de A e Z abaixo são: A=4; B=32; C=12; D=2 e I=8.

- | | |
|---|---------------------------------|
| a. $B + 21 \text{ div } A \text{ div } I$ | c. $I \bmod 5 - A ** (55 - 21)$ |
| b. $C + 6 ** 2 - (21 + A)$ | d. $A * B \text{ div } C + D$ |

- 10) Identifique o tipo de cada expressão vista a seguir:

- | | |
|-------------------------------------|--|
| a. $C + 3 ** 3 * (4 - 7)$ | d. "Júnior" + "Rosa" |
| b. $C + 3 ** 3 > (4 - 7)$ | e. $V \text{ e } F \text{ ou } \neg(V \text{ ou } F)$ |
| c. $12 ** 3 \geq 1200 \text{ e } F$ | f. $V \text{ e } F \text{ ou } \neg(V \text{ ou } F) \text{ e } (2 > 3)$ |

Exercícios da fase “Discuto, Aplico e Reflito”

- 1) Resolva o seguinte problema: “Um feirante vendia queijos em peças. Ao primeiro comprador, ele vendeu a metade das peças que possuía mais meio queijo. Ao segundo, ele vendeu a metade do que restou mais meio queijo. Assim seguiu vendendo até chegar ao sexto e último comprador que comprou a metade do que o feirante possuía mais meio queijo, encerrando as atividades com todos os queijos vendidos. Quantos queijos o vendedor possuía?”.
- 2) O problema a seguir deve ser resolvido através do sistema de equações. Resolva-o construindo as equações e aplicando-as.

Uma pessoa falou com a outra:

– Se você me der R\$1,00, eu terei o dobro do que você tem.

Então o outro disse:

– Se você me der R\$1,00, teremos a mesma quantidade.

Quanto tinha cada um?

- 3) O problema a seguir deve ser resolvido através do sistema de equações. Resolva-o construindo as equações e aplicando-as.

Três casais foram fazer compras em uma feira de exposição. João, José e Juca e suas respectivas esposas: Maria, Marlene e Mara. Quem está casado com quem, se sabemos que cada uma dessas seis pessoas pagou por cada objeto comprado o mesmo número (em R\$) que o número de objetos comprados. Cada homem gastou R\$48,00 a mais que a sua mulher. Além disso, João comprou 9 objetos a mais do que Marlene e José comprou 7 objetos a mais do que Maria. “Sugestão: Considere que, para um certo casal, o homem comprou H objetos e a mulher M objetos”.

- 4) O problema a seguir deve ser resolvido através do sistema de equações. Resolva-o construindo as equações e aplicando-as.

Tenho 26 descendentes, entre filhos, netos e bisnetos. Se eu tivesse mais dois bisnetos, o número destes seria igual ao dos filhos. Quanto aos netos, seu número é 4 vezes o dos bisnetos. Quantos são os meus filhos, netos e bisnetos?

- 5) O problema a seguir deve ser resolvido através do sistema de equações. Resolva-o construindo as equações e aplicando-as.

O tempo que um carro gasta para percorrer o caminho de A e D é $\frac{2}{3}$ do tempo que o mesmo carro leva para voltar de D a A. O carro nº 1 fez o trajeto de A a D em 8 horas e voltou de D a C em 2 horas. O carro nº 2 fez o trajeto de D a A em 13 horas e meia e voltou de A a B em 3 horas. O carro nº 3 fez o trajeto de B a C e voltou de C a B em 7 horas e meia. Qual destes carros desenvolveu maior velocidade?

- 6) O problema a seguir deve ser resolvido através do sistema de equações. Resolva-o construindo as equações e aplicando-as.

Cinco ladrões furtaram, em uma noite, uma coleção de moedas antigas. Cada um dos quatro primeiros ladrões pegou a metade das moedas que encontrou mais duas, o quinto ladrão não levou nenhuma, pois quando chegou para realizar o furto, não havia mais moedas. Qual era o número inicial de moedas?

- 7) O problema a seguir deve ser resolvido através do sistema de equações com operadores relacionais. Resolva-o construindo a equação necessária e deduzindo o resultado através dos valores.

Maria, Júlia, Elisa, Lúcia, e Noêmia foram comprar seus livros escolares, cujos preços variavam de 10, 15, 20 e 25 reais. Cada uma delas comprou um livro de matéria diferente. O livro de Português custou menos do que o de Ciências e mais do que o de História. O livro de Matemática custou mais do que o de Português e mais do que o de Geografia, que custou menos do que os de Ciências e Português. Noêmia gastou mais do que Maria e Julia juntas. Lucia gastou mais do que Elisa. O livro que Júlia comprou custou menos do que o de Matemática, mas não do que o de História. Que livro cada menina comprou e quanto pagou por ele?

- 8) O problema a seguir deve ser resolvido através do sistema de equações. Resolva-o construindo as equações e aplicando-as.

Pedrinho saiu de sua casa e foi para o campinho de futebol do seu bairro a uma velocidade de três quilômetros por hora. Quando chegou lá, lembrou que estava de castigo e voltou para casa correndo duas vezes mais depressa. A mãe de Pedrinho não notou sua ausência, pois este ficou fora de casa por apenas quinze minutos. Qual a distância, em quilômetros, da casa de Pedrinho até o campinho de futebol?

- 9) O problema a seguir deve ser resolvido através do sistema de equações. Resolva-o construindo as equações e aplicando-as.

Renata fez acionar o alarme ao passar na porta de segurança de um aeroporto. Ela levava um porta-moedas com 24 moedas, algumas moedas de R\$ 0,50, 5 moedas de R\$ 0,10 e outras de R\$ 1,00, perfazendo um valor total de R\$ 18,00. Quantas moedas de cada tipo existia no porta-moedas?

- 10) O problema a seguir deve ser resolvido através do sistema de equações. Resolva-o construindo as equações e aplicando-as.

Em um quintal existem porcos, avestruz e galinhas, formando um total de 60 cabeças e 180 pés. Quantos são os animais de duas patas e quantos são os de quatro patas?

Exercícios da fase “Prático, Faço e Crio”

- 1) Represente a expressão a seguir na forma computacional.

$$\frac{x^4 - \sqrt[4]{\frac{x-3y+a}{|2x-y|}} - 3}{x+1} - 4x - y^5$$

- 2) Represente a expressão a seguir na forma computacional.

$$2\frac{x}{5} - \sqrt{x^3 - 4y + 10} + \frac{\sqrt{x-y+|x^5-y^3|}}{x-y+5x^4} + (x-y + \sqrt[4]{x+y})^3$$

- 3) Represente a expressão a seguir na forma computacional.

$$-x^5 + \frac{x-1}{3x+|y-x|} + \sqrt{x+\sqrt[3]{y+2}} + \frac{\sqrt{|x-y|+1}}{\sqrt{x+1}} - 3$$

- 4) Represente a expressão a seguir na forma computacional.

$$\frac{x+y^4-3x^3+\frac{2x-y}{3}}{3\sqrt{x^5+y-3x+\frac{x+1}{2}}} + \sqrt{x+y-2} - \frac{\sqrt[4]{2x-y^5}}{3x^5+\frac{x+3}{2}} + 1$$

- 5) Crie um problema matemático (com duas incógnitas) para ser resolvido com sistema de equações (parecido com os problemas demonstrados em sala de aula).
- 6) Crie um problema matemático (com três incógnitas) para ser resolvido com sistema de equações (parecido com os problemas demonstrados em sala de aula).
- 7) Crie um problema matemático (com quatro incógnitas) para ser resolvido com sistema de equações (parecido com os problemas demonstrados em sala de aula).
- 8) Otimize a equação a seguir através da simplificação de termos.

$$Y(X + 3Y) - (Y^2 + 5XY)$$

- 9) Otimize a equação a seguir através da simplificação de termos.

$$2X^2 + 3XY - 12X^4 - X^2 + X(X^3 - X + Y) - 3X(Y - 1)$$

- 10) Otimize a equação a seguir através da simplificação de termos.

$$Y^4 + 3X + 12X^3 - 4X^2 + X(X^2 - X + Y) - 2X(Y - 7)$$

A.4. DEFINIR DADOS

Exercícios da fase “Vejo, Escuto e Compreendo”

- 1) Quais são os tipos primitivos de dados das seguintes linguagens de programação?
 - a. C++
 - b. Java
 - c. C# (C-Sharp)
 - d. Pascal
- 2) Quais são as palavras-chave das seguintes linguagens de programação?
 - a. C++
 - b. Java
 - c. C# (C-Sharp)
 - d. Pascal
- 3) Quais são as instruções para transformar um inteiro em caractere e vice-versa nas seguintes linguagens de programação?
 - a. C++
 - b. Java
 - c. C# (C-Sharp)
 - d. Pascal
- 4) Faça uma busca na Internet (ou em livros de programação) à procura de pelo menos duas regras básicas para a construção de bons identificadores.
- 5) Descreva as prováveis variáveis necessárias à construção de um algoritmo para calcular as raízes de uma equação do segundo grau.
- 6) Descreva as prováveis variáveis necessárias à construção de um algoritmo para calcular a idade média dos alunos de uma turma.
- 7) Descreva as prováveis variáveis e constantes necessárias à construção de um algoritmo para calcular a média ponderada entre três números.

- 8) Descreva o tipo das prováveis variáveis e constantes necessárias à construção de um algoritmo para calcular:
- a. Volume do Cubo
 - b. Volume da Esfera
 - c. Área do Triângulo
 - d. Área de um Hexágono
- 9) Descreva as prováveis variáveis e constantes necessárias à construção de um algoritmo para calcular:
- a. Volume do Cubo
 - b. Volume da Esfera
 - c. Área do Triângulo
 - d. Área de um Hexágono
- 10) Descreva 5 exemplos de problemas (da física, da química ou de outra ciência) em que o uso de constante se faz necessário.

Exercícios da fase “Discuto, Aplico e Reflito”

- 1) Descreva as prováveis variáveis e constantes necessárias à construção de um algoritmo para calcular o Movimento Retilíneo Uniformemente (**MRU**) como o exemplo a seguir.

Dois carros, A e B, se deslocam numa pista retilínea, ambos no mesmo sentido e com velocidades constantes. O carro que está na frente desenvolve 72 km/h e o que está atrás desenvolve 126 km/h. Num certo instante, a distância entre eles é de 225 m.

- a. Quanto tempo o carro A gasta para alcançar o carro B ?
- b. Que distância o carro que está atrás precisa percorrer para alcançar o que está na frente ?

- 2) Descreva as prováveis variáveis e constantes necessárias à construção de um algoritmo para calcular a **Força** como o exemplo a seguir.

Uma bola de massa 0,40kg é lançada contra uma parede. Ao atingi-la, a bola está se movendo horizontalmente para a direita com velocidade escalar de 15m/s, sendo rebatida horizontalmente para a esquerda com velocidade escalar de 10m/s. Se o tempo de colisão é de $5,0 \times 10^{-3}$ s, qual a intensidade da força média sobre a bola em **Newtons**?

- 3) Descreva as prováveis variáveis e constantes necessárias à construção de um algoritmo para calcular a **Dilatação** de um elemento como o exemplo a seguir.

A dilatação térmica dos sólidos é um fenômeno importante em diversas aplicações de engenharia, como construções de pontes, prédios e estradas de ferro. Considere o fato dos trilhos de trem serem construídos à base de aço, cujo coeficiente de dilatação é $\alpha = 11 \times 10^{-6} \text{ }^{\circ}\text{C}^{-1}$. Se a 10°C o comprimento de um trilho é de 30m, de quanto aumentaria o seu comprimento se a temperatura aumentasse para 40°C ?

- 4) Descreva as prováveis variáveis e constantes necessárias à construção de um algoritmo para calcular a **Diferença de Potencial** como o exemplo a seguir.

Um pássaro pousa em um dos fios de uma linha de transmissão de energia elétrica. O fio conduz uma corrente elétrica $i = 1.000 \text{ A}$ e sua resistência, por unidade de comprimento, é de $5,0 \times 10^{-5} \text{ W/m}$. A distância que separa os pés do pássaro, ao longo do fio, é de 6,0 cm. A diferença de potencial, em milivolts (mV), entre os seus pés é: “?”

- 5) Descreva as prováveis variáveis e constantes necessárias à construção de um algoritmo para calcular a **Massa** como o exemplo a seguir.

Uma fábrica de produtos metalúrgicos do Distrito Industrial de Fortaleza consome, por mês, cerca de $2,0 \times 10^6 \text{ kWh}$ de energia elétrica ($1 \text{ kWh} = 3,6 \times 10^6 \text{ J}$). Suponha que essa fábrica possui uma usina capaz de converter diretamente massa em energia elétrica, de acordo com a relação de Einstein, $E = m_0 c^2$. Nesse caso, a massa necessária para suprir a energia requerida pela fábrica, durante um mês, é, em gramas: “?”

- 6) Descreva as prováveis variáveis e constantes e os passos necessários à construção de um algoritmo que valida um CPF (Considere que o CPF está armazenado em um conjunto de 11 caracteres).
- 7) Descreva as prováveis variáveis e constantes e os passos necessários à construção de um algoritmo que valida um CNPJ (Considere que o CNPJ está armazenado em um conjunto de 14 caracteres).

- 8) Descreva as prováveis variáveis e constantes necessárias à construção de um algoritmo para calcular o **Índice de refração e Espessura** como o exemplo a seguir.

Um feixe de luz de cor laranja, cujo comprimento de onda, no vácuo, é $\lambda = 600$ nm ($1 \text{ nm} = 1 \times 10^{-9} \text{ m}$), atravessa um bloco de cristal de espessura **L**. Essa luz demora apenas um tempo $\Delta t = 2 \text{ ns}$ para atravessar o cristal e seu comprimento de onda ali fica reduzido a $\lambda_n = 400 \text{ nm}$. O índice de refração **n** do cristal e sua espessura **L** têm valores dados, respectivamente, por: “?”

- 9) Descreva 5 situações em que o uso de dados do tipo inteiro é obrigatório e outras 5 situações em que é opcional.
- 10) Descreva 5 situações em que o uso de dados do tipo real é obrigatório.

Exercícios da fase “Prático, Faço e Crio”

- 1) Dê 1 exemplo e 1 contra-exemplo de identificador significativo para cada provável variável e constante necessárias à construção de um algoritmo para calcular o Movimento Retilíneo Uniformemente (**MRU**) como o exemplo a seguir.

Dois carros, A e B, se deslocam numa pista retilínea, ambos no mesmo sentido e com velocidades constantes. O carro que está na frente desenvolve 72 km/h e o que está atrás desenvolve 126 km/h . Num certo instante, a distância entre eles é de 225 m .

- Quanto tempo o carro A gasta para alcançar o carro B?
- Que distância o carro que está atrás precisa percorrer para alcançar o que está na frente?

- 2) Dê 1 exemplo e 1 contra-exemplo de identificador significativo para cada provável variável e constante necessárias à construção de um algoritmo para calcular a **Força** como o exemplo a seguir.

Uma bola de massa $0,40 \text{ kg}$ é lançada contra uma parede. Ao atingi-la, a bola está se movendo horizontalmente para a direita com velocidade escalar de 15 m/s , sendo rebatida horizontalmente para a esquerda com velocidade escalar de 10 m/s . Se o tempo de colisão é de $5,0 \times 10^{-3} \text{ s}$, qual a intensidade da força média sobre a bola em **Newtons**?

- 3) Dê 1 exemplo e 1 contra-exemplo de identificador significativo para cada provável variável e constante necessárias à construção de um algoritmo para calcular a **Dilatação** de um elemento como o exemplo a seguir.

A dilatação térmica dos sólidos é um fenômeno importante em diversas aplicações de engenharia, como construções de pontes, prédios e estradas de ferro. Considere o caso dos trilhos de trem serem construídos à base de aço, cujo coeficiente de dilatação é $\alpha = 11 \times 10^{-6} \text{ }^\circ\text{C}^{-1}$. Se a 10°C o comprimento de um trilho é de 30m, de quanto aumentaria o seu comprimento se a temperatura aumentasse para 40°C ?

- 4) Dê 1 exemplo e 1 contra-exemplo de identificador significativo para cada provável variável e constante necessárias à construção de um algoritmo para calcular a **Diferença de Potencial** como o exemplo a seguir.

Um pássaro pousa em um dos fios de uma linha de transmissão de energia elétrica. O fio conduz uma corrente elétrica $i = 1.000 \text{ A}$ e sua resistência, por unidade de comprimento, é de $5,0 \times 10^{-5} \text{ W/m}$. A distância que separa os pés do pássaro, ao longo do fio, é de 6,0 cm. A diferença de potencial, em milivolts (mV), entre os seus pés é: “?”

- 5) Dê 1 exemplo e 1 contra-exemplo de identificador significativo para cada provável variável e constante necessárias à construção de um algoritmo para calcular a **Massa** como o exemplo a seguir.

Uma fábrica de produtos metalúrgicos do Distrito Industrial de Fortaleza consome, por mês, cerca de $2,0 \times 10^6 \text{ kWh}$ de energia elétrica ($1 \text{ kWh} = 3,6 \times 10^6 \text{ J}$). Suponha que essa fábrica possui uma usina capaz de converter diretamente massa em energia elétrica, de acordo com a relação de Einstein, $E = m_0 c^2$. Nesse caso, a massa necessária para suprir a energia requerida pela fábrica, durante um mês. é. em gramas: “?”

- 6) Dê 1 exemplo e 1 contra-exemplo de identificador significativo para cada provável variável e constante necessárias à construção de um algoritmo que valida um CPF (Considere que o CPF está armazenado em um conjunto de 11 caracteres).

- 7) Dê 1 exemplo e 1 contra-exemplo de identificador significativo para cada provável variável e constante necessárias à construção de um algoritmo que valida um CNPJ (Considere que o CNPJ está armazenado em um conjunto de 14 caracteres).
- 8) Dê 1 exemplo e 1 contra-exemplo de identificador significativo para cada provável variável e constante necessárias à construção de um algoritmo para calcular o **Índice de refração e Espessura** como o exemplo a seguir.

Um feixe de luz de cor laranja, cujo comprimento de onda, no vácuo, é $\lambda = 600$ nm ($1 \text{ nm} = 1 \times 10^{-9} \text{ m}$), atravessa um bloco de cristal de espessura **L**. Essa luz demora apenas um tempo $Dt = 2 \text{ ns}$ para atravessar o cristal e seu comprimento de onda ali fica reduzido a $\lambda_n = 400 \text{ nm}$. O índice de refração **n** do cristal e sua espessura **L** têm valores dados, respectivamente, por: “?”

- 9) Imagine o problema do cálculo da folha de pagamento de uma empresa. Assinale os identificadores considerados bons candidatos a ocorrerem (como constantes ou variáveis) no algoritmo construído para este problema. Justifique sua resposta.
- | | |
|-------------------------|--------------------------|
| a. Alíquota_de_IR_15 | k. A_IR275 |
| b. Alíquota_de_IR_275 | l. Q_F |
| c. Qtd_h_e | m. Quantidade_faltas |
| d. Qtd_horas_extras_50 | n. Q_H_T |
| e. Qtd_horas_extras_100 | o. Quantidade_horas_trab |
| f. Sal | p. Quantidade_horas |
| g. Salário_bruto | q. Alíquota_INSS_11 |
| h. Salário_liquido | r. A_INSS_15 |
| i. Sal_L | s. Ind_Salub |
| j. A_IR15 | t. Índice_salubridade |

10) Imagine o problema do controle de caixa de um posto de combustível. O posto deve fazer o controle de todos os produtos vendidos (de acordo com a quantidade vendida e o preço unitário) e todos os serviços prestados (de acordo com o valor do serviço). Assinale os identificadores considerados bons candidatos a ocorrerem (como constantes ou variáveis) no algoritmo construído para este problema. Justifique sua resposta.

- | | |
|------------------------------|-----------------------------|
| a. Valor_Litro_Gasolina | k. Quantidade_Duchas |
| b. Val_Diesel | l. Q_Ducha |
| c. ValorAlcool | m. Troc_Ol |
| d. ValorLitroAlcoolAditivado | n. QuantidadeTrocaDeOleo |
| e. Val_Lt_Alcool_Comum | o. Qnt_Lt_Oleo |
| f. Quant_Diesel | p. Quantidade_Litros_Oleo |
| g. Qnt_Ab_Diesel | q. Tur |
| h. Quantidade_Abastecida_G | r. Turno_trabalho |
| i. Ducha | s. Qnt_Veic_abastecidos_mes |
| j. Valor_Ducha | t. TotVeicAbastecidosAno |

A.5. CONSTRUIR TIPOS ABSTRATOS DE DADOS

Exercícios da fase “Vejo, Escuto e Compreendo”

- 1) Construa (com nomes significativos e tamanho ideal) vetores para armazenar:
- O nome de um aluno (conjunto de 40 caracteres).
 - O campo “Endereço” de um formulário qualquer (conjunto de 60 caracteres).
 - A nota de matemática de uma turma de 50 alunos (conjunto de valores reais).
 - Os salários de 1000 funcionários (conjunto de valores reais).
 - As idades dos 185 funcionários da escola para calcular a média de idade (conjunto de valores inteiros).
 - Os telefones (sem prefixo e DDD) de todos os 478 clientes de uma loja (conjunto de valores reais ou inteiros estendidos).

- 2) Construa o tipo de dado “Cliente” que possua os seguintes atributos: Nome, Endereço, Bairro, Telefone, CEP, Cidade.
- 3) Construa o tipo de dado “Veículo” que possua os seguintes atributos: Modelo, Marca, AnoFabricação, ModeloFabricação, Placa. Escolha o tamanho e o tipo de cada atributo à sua vontade.
- 4) Construa o tipo de dado “Aluno” que possua os seguintes atributos: Matrícula, Nome, Idade, CPF, Curso. Escolha o tamanho e o tipo de cada atributo à sua vontade.
- 5) Construa um algoritmo para ler e mostrar os dados de um cliente cujo tipo de dado possua os seguintes atributos: Nome, Idade, RG. Escolha o tamanho e o tipo de cada atributo à sua vontade.
- 6) Construa um algoritmo para ler e mostrar os dados de um fornecedor cujo tipo de dado possua os seguintes atributos: Razão_Social, Nome_Fantasia, CNPJ, Endereço, RamoComercial. Escolha o tamanho e o tipo de cada atributo à sua vontade.
- 7) Construa um algoritmo para ler e mostrar de um contracheque de um funcionário cujo tipo de dado possua os seguintes atributos: Nome, CódigoMatrículaFuncionário, SalárioBruto, SalárioLíquido, IRRFDescontar, INSSDescontar, HorasTrabalhadas. Escolha o tamanho e o tipo de cada atributo à sua vontade.
- 8) Construa um algoritmo para ler os salários de 3 funcionários (em um vetor). Realize neles um reajuste de 35% e mostre os salários reajustados ao final. Os salários devem ser representados em um tipo real.
- 9) Construa um algoritmo para ler 1 número (armazenando-o na primeira posição de um vetor de 5 inteiros). Em seguida, calcule o 2º número somando o primeiro número do vetor ao valor da posição deste novo número no vetor (ou seja, o valor 2). Em seguida, calcule o 3º número somando o segundo número do vetor ao valor da posição deste novo número no vetor (ou seja, o valor 3) e assim por diante. Depois de calculados o 4º e o 5º números do vetor, você deverá mostrá-los no sentido do final para o início do vetor.
- 10) Construa um algoritmo com uma matriz de inteiros 3 X 3. Leia os valores de cada elemento da matriz. Calcule e mostre o determinante desta matriz.

Exercícios da fase “Discuto, Aplico e Reflito”

- 1) Analise a corretude do algoritmo abaixo e corrija os erros encontrados. Justifique os erros que apontar e explique o efeito colateral que poderia ser causado por eles.

Algoritmo "PROGRAMA MÉDIA"

// Função: LÊ 4 NÚMEROS DO TECLADO E CALCULA A MÉDIA.

// Para calcular a média entre 4 números você deve somá-los e dividir a soma por 4

var

N: vetor [1..2] de inteiro

SOMA: INTEIRO

MEDIA: CARACTERE

// Por que a soma e a média não são vetores? Porque a soma e a média são únicas

// A soma contém um valor acumulado de valores

// A média contém um valor calculado

inicio

// Seção de Comandos de entrada

Escreva("DIGITE O 1º NÚMERO:")

Leia (N[1])

Escreva("DIGITE O 2º NÚMERO:")

Leia (N[2])

Escreva("DIGITE O 3º NÚMERO:")

Leia (N[3])

Escreva("DIGITE O 4º NÚMERO:")

Leia (N[4])

// Seção de cálculo (Processamento)

SOMA <- 0

SOMA <- N[1] + N[2] + N[3] + N[4]

MEDIA <- SOMA / 4

// Seção de saída (exibição) de dados

Escreva("A MÉDIA ARITMÉTICA É: ", SOMA)

Fimalgoritmo

- 2) Analise a corretude do algoritmo abaixo e corrija os erros encontrados. Justifique os erros que apontar e explique o efeito colateral que poderia ser causado por eles. Analise também a qualidade (significância para o contexto do problema) dos comentários agregados ao algoritmo.

```
Algoritmo "CÁLCULO DA ÁREA DE 3 TRIÂNGULOS"
// Função : Calcular a área de 3 triângulos.

var
    // Variáveis do tipo real
    Area_T1, Base_T1, Altura_T1: real
    Area_T2, Base_T2, Altura_T2, Area_T3, Base_T3, Altura_T3: real

inicio
    // A área de um triângulo é calculada pela seguinte fórmula:
    // Área = (Base * Altura) / 2

    // Seção de Comandos de entrada
    escreva("Digite o tamanho da base do 1º triângulo: ")
    leia(Base_T1)
    escreva("Digite a altura do 1º triângulo: ")
    leia(Altura_T1)

    escreva("Digite o tamanho da base do 2º triângulo: ")
    leia(Base_T2)
    escreva("Digite a altura do 2º triângulo: ")
    leia(Altura_T2)

    escreva("Digite o tamanho da base do 3º triângulo: ")
    leia(Base_T2)
    escreva("Digite a altura do 2º triângulo: ")
    leia(Altura_T3)

    // Seção de cálculo (Processamento)
    // Calculando
    Area_T1 <- (Base_T1*Altura_T1)/2
    Area_T2 <- (Base_T2*Altura_T2)/2
    Area_T3 <- (Base_T3*Altura_T3)/3

    // Seção de saída (exibição) de dados
    escreva("A área do 1º triângulo é: ", Area)
    escreva("A área do 2º triângulo é: ", Area2)
    escreva("A área do 3º triângulo é: ", Area_T3)

    // Fim do algoritmo
Fimalgoritmo
```

- 3) Analise a corretude do algoritmo abaixo e corrija os erros encontrados. Justifique os erros que apontar e explique o efeito colateral que poderia ser causado por eles. Analise a qualidade (significância para o contexto do problema) dos comentários agregados ao algoritmo. Analise também a significância dos identificadores encontrados.

Algoritmo "LENDO OS DADOS DE UM FUNCIONÁRIO"

```
// Função : Ler e mostrar os dados de um funcionário

var
    // Seção de Criação de Tipos Novos
    Funcionario: Registro
        NOME: vetor [1..30] de caractere
        MATRICULA: inteiro
        SALARIO: caractere
    FimRegistro

    // Variável do tipo FUNCIONARIO
    TRAB: Funcionario
inicio
    // Seção de Comandos de entrada
    escreva("Digite o nome do funcionário: ")
    leia(TRAB.NOME)
    escreva("Digite a matrícula do funcionário: ")
    leia(TRAB.ENDERECO)
    escreva("Digite o salário do funcionário: ")
    leia(TRAB.SALARIO)
Fimalgoritmo
```

- 4) Analise a corretude do algoritmo abaixo e corrija os erros encontrados. Justifique os erros que apontar e explique o efeito colateral que poderia ser causado por eles.

Algoritmo "CALCULANDO O RESTO DA DIVISÃO"

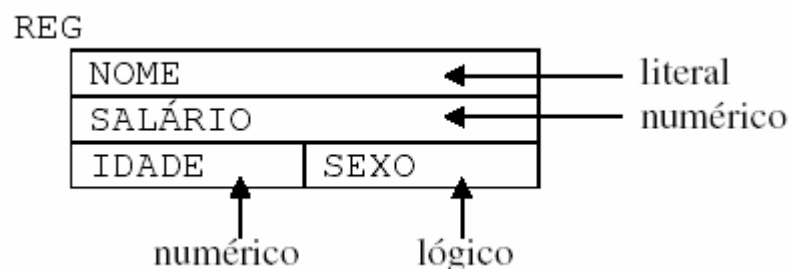
```
// Função : Ler um número e calcular e mostrar o resto da divisão entre o número
// lido e o número 5.

var
    NUM, QUOC, RESTO: inteiro
inicio
    // Seção de Comandos de entrada
    escreva("Digite um número qualquer sem casa decimal: ")
    leia(NUM)
    // Seção de cálculo (Processamento)
    QUOC = NUM / 5
    RESTO = NUM - (QUOC * 5)
    // Seção de saída (exibição) de dados
    escreva("O resto da divisão entre ", NUM, " e 5 é: ", QUOC)
Fimalgoritmo
```

- 5) Imagine um sistema de controle de Pacientes de uma clinica médica. Construa o tipo de dado “PACIENTE” que possua os atributos que você julgar importantes para controlar consultas (por médico e especialidade médica) e possíveis doenças. Construa também um algoritmo para ler os dados de 3 pacientes. Ao final, mostre a idade média dos pacientes.
- 6) Imagine um sistema de controle de Locadora de Fitas. Construa o tipo de dado “FILME” que possua os atributos que você julgar importantes para controlar empréstimo, cadastro por gênero (romance, ação, etc.) e categoria (lançamento, catálogo, etc.) do filme. Construa também um algoritmo para ler os dados de 2 filmes. Faça um reajuste de 18% no valor de locação de cada filme. Ao final, mostre o valor de locação (corrigido) de cada filme.
- 7) Imagine um sistema de controle de vendas de uma loja de roupas. Construa um tipo de dados para armazenar dados relativos à venda como, por exemplo, NomeComprador, CPF, Produto, QuantidadeComprada e outros 5 atributos abstraídos por você. Construa um algoritmo para ler e mostrar todas as 6 vendas registradas no sábado passado. Calcule e mostre o valor total destas transações de venda.
- 8) Construa um algoritmo para ler uma matriz 6 x 6 de inteiros e calcular e mostrar o determinante. Responda à seguinte questão: “Como construir um programa para calcular o determinante de uma matriz $N \times N$? Ou seja, como construir um algoritmo para calcular o determinante de uma matriz quadrática de tamanho genérico?”.
- 9) Descreva o tamanho e o tipo de um vetor armazenar (crie o vetor):
 - A.8. Uma placa de automóvel.
 - A.8. Um RG
 - A.8. O número do chassi de um automóvel.
 - A.8. O número do Renavam de um automóvel.
 - A.8. O telefone de uma pessoa (COD_Internacional + Oeradora + DDD + NUM)
- 10) Descreva os atributos que um veículo automotor, um veículo aéreo e um veículo marítimo (especificamente navio de carga ou passageiros) tem em comum. Construa um tipo abstrato de dados para cada um deles. Construa também um algoritmo para ler os dados de cada um dos veículos. Imagine que todos os três veículos podem conter N passageiros. Calcule e mostre a quantidade total de passageiros existentes somando-se os três veículos.

Exercícios da fase “Pratico, Faça e Crio”

- 1) Faça um algoritmo para cadastrar as 4 notas bimestrais de 5 alunos em uma determinada matéria usando uma matriz 5 x 4. Ao final, mostrar a nota total de cada aluno naquela matéria, a nota média dos alunos por bimestre e a nota média final.
- 2) Faça um algoritmo para armazenar os elementos de uma matriz 3 x 3 x 4 e armazenar todos os seus elementos em um vetor. Calcular e mostrar a média dos elementos do novo vetor.
- 3) Imagine que a velocidade máxima permitida em um cruzamento é de 60 km/h, vigiado por um semáforo equipado com radar e câmera de vigilância que fotografa todo veículo que cruza o sinal (considere uma via de um único sentido). Depois que o sinal fotografa a placa e registra a velocidade de uma quantia de 10 veículos, deve ser calculada a multa para todos os veículos infratores (aqueles que trafegavam acima dos 60 km/h). Para cada quilometro por hora acima da velocidade máxima permitida, deverá ser adicionado R\$ 5,00 à multa. Assim, se o condutor ultrapassar o semáforo a 77 km/h, deverá pagar uma multa de $17 * R\$ 5,00$. Construa uma estrutura de dados para armazenar estas informações no cruzamento e o algoritmo para lê-los e calcular e mostrar o valor pago em multas e o valor médio pago em multas.
- 4) Desenvolva um algoritmo que armazene em um vetor e imprima os 8 primeiros termos da série de Fibonacci, lembrando que esta série é definida da seguinte forma:
 - Termo 1 = 1
 - Termo 2 = 1
 - Termo (x + 1) = termo (x) + termo (x - 1)
- 5) Definir e declarar o registro cuja representação gráfica é dada a seguir. Construir um algoritmo para cadastrar 5 funcionários. Calcule e mostre quanto a empresa gasta por mês com o pagamento de todos os salários.



- 6) Defina um tipo abstrato de dados e um vetor de três atletas com os seguintes atributos: nome, sobrenome, ano de nascimento e altura (em metros). Construa um algoritmo como operações para a iniciação desses atributos (através de digitação). Adicione uma operação que retorne a idade aproximada de todos os atletas de acordo com um determinado ano de entrada. Adicione outra operação que retorne a altura em centímetros de todos os atletas armazenados no vetor. Não se esqueça de incluir comentários no código.
- 7) Em uma fábrica trabalham homens e mulheres que recebem o salário mínimo (R\$510,00) + adicional por serviço prestado. A fábrica possui 5 operários. Faça um algoritmo que leia, para cada operário: o seu número de matrícula (inteiro), o valor do adicional por serviço prestado e seu sexo (1 para masculino ou 2 para feminino). Os dados devem ser armazenados em 3 vetores: vetNumOp, vetAdicServ e vetSexo respectivamente. O algoritmo deve calcular os salários dos funcionários, armazená-los em um quarto vetor (vetSalarios) e mostrar um relatório que contenha o número do operário, o sexo e o seu salário no mês. O programa deve mostrar também o total da folha de pagamento da fábrica e o salário médio.
- 8) Construa um algoritmo que armazene elementos em uma matriz 2 x 3 (como exemplo a seguir). Em seguida, crie outra matriz que deverá ser transposta em relação à primeira. Mostre todos os elementos da matriz transposta com suas respectivas linhas e colunas.

$$A = \begin{pmatrix} 5 & 7 & 3 \\ 1 & 6 & 1 \end{pmatrix} \Leftrightarrow A^t = \begin{pmatrix} 5 & 1 \\ 7 & 6 \\ 3 & 1 \end{pmatrix}$$

- 9) Construa um algoritmo que armazene a matriz a seguir. Em seguida, calcule e mostre a sua matriz inversa.

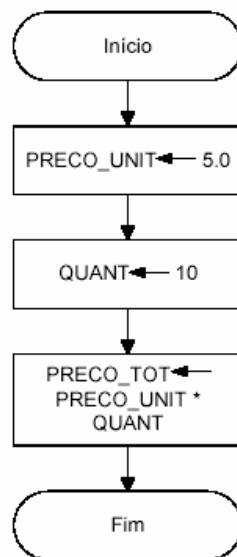
$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 4 & 3 \end{bmatrix}$$

- 10) Construa um algoritmo que leia uma matriz 3 x 2 e outra 2 x 2. Em seguida, calcule o produto das duas matrizes em uma terceira matriz. Mostre a matriz resultante e os seus respectivos índices (linha e coluna).

A.6. CONHECER OS TIPOS DE ESTRUTURAS DE CONTROLE E ENTENDER AS ESTRUTURAS DE SEQUÊNCIA

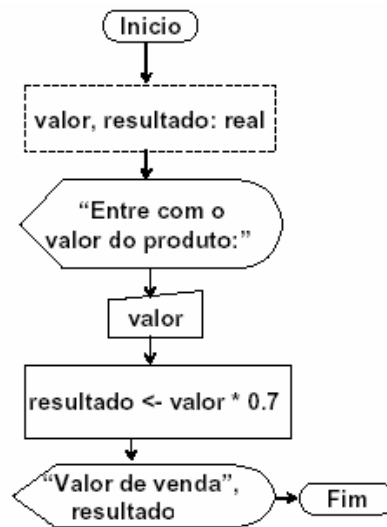
Exercícios da fase “Vejo, Escuto e Compreendo”

- 1) Descreva todas as instruções sequenciais que você se lembra de ter utilizado (inclusive as expressões algébricas) até o presente momento.
- 2) Faça um resumo de 5 páginas (manuscrito) sobre os conceitos de estruturas de controle (sequenciais, de seleção ou condicionais, de repetição).
- 3) Faça um algoritmo que leia um dado valor em graus Fahrenheit e converta-o para graus centígrados. Construa um fluxograma correspondente.
- 4) Faça o algoritmo correspondente ao fluxograma abaixo. Adicione comentários significativos.



- 5) Faça um algoritmo que receba um valor de entrada que representa o tamanho da diagonal em polegadas de uma televisão LCD e calcule e mostre quantos centímetros mede esta diagonal. Construa um fluxograma correspondente.

- 6) Faça o algoritmo correspondente ao fluxograma abaixo. Adicione comentários significativos.



- 7) Faça um algoritmo que receba um valor de entrada que representa a altura de um avião em pés e calcule e mostre a quantos metros de altura se encontra o avião. Construa um fluxograma correspondente.
- 8) Faça um algoritmo que leia a idade de uma pessoa (expressa em anos, meses e dias) em um vetor com três números inteiros. O primeiro elemento do vetor armazena a idade em anos. O segundo elemento armazena a quantidade de meses vividos no último ano e o terceiro elemento armazena a quantidade de dias vividos no último mês. Considere um ano de 365 dias (ignore a existência de anos bissextos). Calcule e mostre a quantidade de dias vividos e de horas vividas até então. Construa um fluxograma correspondente.
- 9) O custo ao consumidor de um carro novo é a soma do custo de fábrica com a percentagem do distribuidor (aplicados ao custo de fábrica) e dos impostos (aplicados ao custo do distribuidor). Supondo que a percentagem do distribuidor seja de 28% e os impostos de 45%, escrever um algoritmo que leia o custo de fábrica de um carro e escreva o custo ao consumidor. Calcule e mostre qual o percentual final de aumento teve o preço do carro até chegar ao consumidor. Construa um fluxograma correspondente.
- 10) Escreva um algoritmo e o seu fluxograma correspondente para calcular o consumo médio de um automóvel (medido em Km/l), dado que são conhecidos (digitados) a distância total percorrida e o volume de combustível consumido para percorrê-la (medido em litros).

Exercícios da fase “Discuto, Aplico e Reflito”

- 1) Utilizando o seguinte trecho de algoritmo, explique o que está acontecendo em cada linha e qual é o resultado de cada ação executada:

```
.  
.   
X, Y: inteiro  
Z: real  
Leia(X)  
Escreva(X, “elevado ao cubo =”, X ** 3)  
Leia(Y)  
Escreva(X + Y)  
Z = X / Y  
Escreva(Z)  
Z = Z + 1  
X = (Y + X) mod 2  
Escreva(X)  
.   
.
```

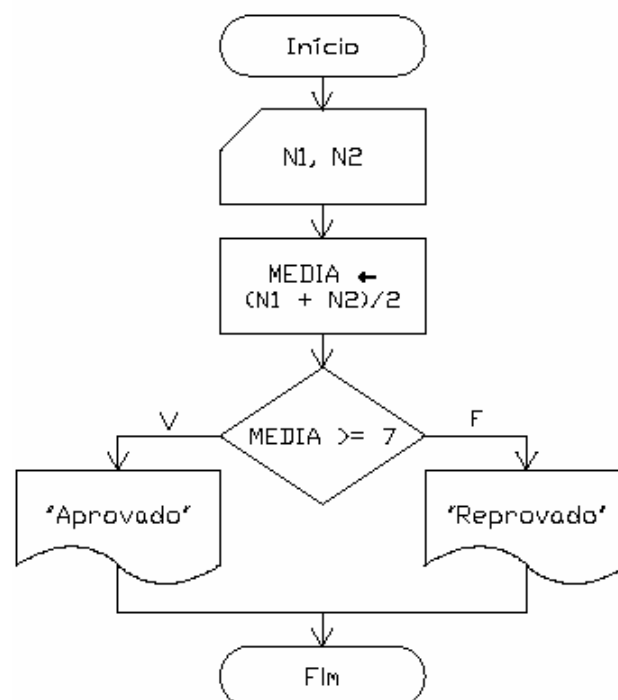
- 2) Analise o algoritmo abaixo e indique qual o seu propósito. Não seria mais fácil compreendê-lo se os identificadores fossem mais significativos e se existissem linhas de comentários elucidativas? Reescreva o algoritmo com identificadores mais significativos em relação ao contexto e adicione comentários onde considerar importante. Construa um fluxograma correspondente.

```
Algoritmo “XXXXXX”  
var  
    X1, X2, X3, X4, Y: real  
inicio  
    escreva(“Digite X1”)  
    leia (X1)  
    escreva(“Digite X2”)  
    leia (X2)  
    escreva(“Digite X3”)  
    leia (X3)  
    escreva(“Digite X4”)  
    leia (X4)  
    Y = (X1 + X2 + X3 + X4) / 4  
    escreva(Y)  
FimAlgoritmo.
```

- 3) Analise o algoritmo abaixo e indique qual o seu propósito. Não seria mais fácil compreendê-lo se os identificadores fossem mais significativos e se existissem linhas de comentários elucidativas? Reescreva o algoritmo com identificadores mais significativos em relação ao contexto e adicione comentários onde considerar importante. Construa um fluxograma correspondente.

```
Algoritmo "YYYYYY"  
  const Z = 3.1416  
  var  
    R, A: real  
  inicio  
    leia(R)  
    A = Z * R ** 2  
    Escreva("XXXXX é: ", A)  
Fimalgoritmo
```

- 4) Tente interpretar e descrever o que ocorre no fluxograma a seguir. Identifique nele uma estrutura que ainda não tenha sido utilizada na sala de aula para desenvolver algoritmos.



- 5) Analise o algoritmo abaixo e indique qual o seu propósito. Não seria mais fácil compreendê-lo se os identificadores fossem mais significativos e se existissem linhas de comentários elucidativas? Reescreva o algoritmo com identificadores mais significativos em relação ao contexto e adicione comentários onde considerar importante. Construa um fluxograma correspondente.

Algoritmo “MMMMM”

var

X1: inteiro

Inicio

X1 = 0

escreva(“O primeiro _____ é: ”, X1)

X1 = X1 + 2

escreva(“O segundo _____ é: ”, X1)

X1 = X1 + 2

escreva(“O terceiro _____ é: ”, X1)

X1 = X1 + 2

escreva(“O quarto _____ é: ”, X1)

X1 = X1 + 2

escreva(“O quinto _____ é: ”, X1)

FimAlgoritmo.

- 6) Analise o algoritmo abaixo e o complete corretamente (adicionando as expressões algébricas que correspondem às linhas de saída de dados posteriores a cada expressão). Não seria mais fácil compreendê-lo se os identificadores fossem mais significativos e se existissem linhas de comentários elucidativas? Reescreva o algoritmo com identificadores mais significativos em relação ao contexto e adicione comentários onde considerar importante. Construa um fluxograma correspondente.

Algoritmo “NNNNNNN”

var

X1, X2, X3, X4: inteiro

Inicio

escreva(“Digite um valor em metros: ”, X1)

X2 = _____

escreva(“Este é o valor correspondente em milímetros: ”, X2)

X3 = _____

escreva(“Este é o valor correspondente em quilômetros: ”, X3)

X4 = _____

escreva(“Este é o valor correspondente em milhas: ”, X4)

FimAlgoritmo.

- 7) Analise o algoritmo abaixo e indique qual o seu propósito. A seguir, divida-o em blocos (circundando-os com uma caneta) de acordo com a funcionalidade das instruções ou conjunto de instruções (Por exemplo, separe o bloco de entrada de dados, o bloco de processamento – em quantas tarefas de processamento existir – e o bloco de saída de dados). Não seria mais fácil compreendê-lo se os identificadores fossem mais significativos e se existissem linhas de comentários elucidativas? Reescreva o algoritmo com identificadores mais significativos em relação ao contexto e adicione comentários onde considerar importante. Construa um fluxograma correspondente.

Algoritmo “OOOOOOO”

var

X1, X2, X3, X4, Y, Z, M, R: real

K1, K2, K3, K4: inteiro

Início

escreva(“Digite X1”)

leia (X1)

escreva(“Digite X2”)

leia (X2)

escreva(“Digite X3”)

leia (X3)

escreva(“Digite X4”)

leia (X4)

escreva(“Digite K1”)

leia (K1)

escreva(“Digite K2”)

leia (K2)

escreva(“Digite K3”)

leia (K3)

escreva(“Digite K4”)

leia (K4)

$Y = (X1 + X2 + X3 + X4)$

$Z = Y / 4$

$R = (X1 * K1 + X2 * K2 + X3 * K3 + X4 * K4)$

$M = R / (K1 + K2 + K3 + K4)$

escreva(M)

escreva(Z)

FimAlgoritmo.

- 8) Construa um algoritmo e seu correspondente fluxograma que, tendo como dados de entrada dois pontos quaisquer do plano, $P(X_1, Y_1)$ e $Q(X_2, Y_2)$, calcule e imprima a distância entre eles. Não se esqueça de comentar o algoritmo com uma explicação detalhada da fórmula (expressão) necessária para resolver o problema.

- 9) Analise o algoritmo abaixo e indique qual o seu propósito (Elabore um enunciado). A seguir, divida-o em blocos (circundando-os com uma caneta – caso seja necessário, reescreva-o antes de separá-lo em blocos) de acordo com a funcionalidade das instruções ou conjunto de instruções (Por exemplo, separe o bloco de entrada de dados, o bloco de processamento – em quantas tarefas de processamento existir – e o bloco de saída de dados). Não seria mais fácil compreendê-lo se os identificadores fossem mais significativos e se existissem linhas de comentários elucidativas? Reescreva o algoritmo com identificadores mais significativos em relação ao contexto e adicione comentários onde considerar importante. Construa um fluxograma correspondente para cada funcionalidade identificada.

Algoritmo “ZZZZZZZZZ”

var

const Z = 3.1416

X1, X2, X3, X4, X5, X6, R1, R2, R3, R4: real

Inicio

escreva(“Digite X1”)

leia (X1)

escreva(“Digite X2”)

leia (X2)

escreva(“Digite X3”)

leia (X3)

escreva(“Digite X4”)

leia (X4)

escreva(“Digite X5”)

leia (X5)

escreva(“Digite X6”)

leia (X6)

R1 = X1 * X2

escreva(R1)

R2 = (X3 * X4) / 2

escreva(R2)

R3 = Z * X5 ** 2

escreva(R3)

R4 = X6 ** 2

escreva(R4)

FimAlgoritmo.

- 10) Construa um algoritmo e seu correspondente fluxograma para ler os valores de dois vetores com 5 inteiros cada um. Crie um terceiro vetor contendo os valores do primeiro e do segundo vetor de maneira intercalada (Por exemplo, o 1º elemento do 3º vetor = ao 1º elemento do 1º vetor, o 2º elemento do 3º vetor = ao 1º elemento do 2º vetor, o 3º elemento do 3º vetor = ao 2º elemento do 1º vetor, e assim sucessivamente).

Exercícios da fase “Prático, Faça e Crio”

- 1) O código a seguir pode ser otimizado? Em caso positivo, reescreva-o mais otimizado possível.

Algoritmo “CCCCC”

var

X: vetor [1..3] de inteiro

Soma: inteiro

Z: real

inicio

Soma = 0

Escreva(“Digite o primeiro número”)

Leia(X[1])

Soma = Soma + X[1]

Escreva(“Digite o segundo número”)

Leia(X[2])

Soma = Soma + X[2]

Escreva(“Digite o terceiro número”)

Leia(X[3])

Soma = Soma + X[3]

Z = Soma / 3

Escreva(“A média é : ”, Z)

fimalgoritmo

- 2) Construa um algoritmo para calcular o segundo lado de um retângulo cuja área mede 30m e a diagonal mede 8m.
- 3) Imagine se não existisse a instrução “raiz()” em algoritmo ou “sqrt ()” nas linguagens de programação. Usando variáveis do tipo real, escreva uma instrução ou conjunto de instruções que poderia(m) ser usada(s) para calcular: (1) a raiz quadrada de um número; (2) a raiz cúbica de um número; e (1) a raiz à quarta de um número;

- 4) Construa um algoritmo para calcular a distância entre dois locais na terra. O algoritmo deverá ler a longitude e latitude de cada um desses locais e mostrar a distância entre eles em linha reta.
- 5) Construa um algoritmo para calcular a área de um polígono hexágono regular cujo lado mede 4m.
- 6) O código a seguir pode ser otimizado? Em caso positivo, reescreva-o mais otimizado possível.

Algoritmo “FFFFFFF”

var

X1, X2, P1, P2: real

Z: real

inicio

Soma = 0

Escreva(“Digite o primeiro número”)

Leia(X1)

Escreva(“Digite o primeiro peso”)

Leia(P1)

Escreva(“Digite o segundo número”)

Leia(X2)

Escreva(“Digite o segundo peso”)

Leia(P2)

$Z = (X1 * P1 + X2 * P2) / (P1 + P2)$

Escreva(“A média ponderada é : ”, Z)

fimalgoritmo

- 7) Qual dos exemplos a seguir possui um código mais eficiente? Qual é mais legível? Justifique.

1) .
.
Escreva(“Digite um número”)
Leia(X[1])
 $P = X[1] * X[1] * X[1]$
Escreva(P)

.
.

2) .
.
Escreva(“Digite um número”)
Leia(X[1])
 $P = X[1] ** 3$
Escreva(P)

.
.

- 8) Qual dos exemplos a seguir possui um código mais eficiente? Qual é mais legível? Justifique.

```
1)  .  
    .  
    Q, X, R: inteiro  
    Escreva("Digite um número")  
    Leia(X)  
    Q = X / 7  
    Q = Q * 7  
    R = X - Q  
    Escreva(R)  
    .  
    .
```

```
2)  .  
    .  
    X, R: inteiro  
    Escreva("Digite um número")  
    Leia(X)  
    R = X mod 7  
    Escreva(R)  
    .  
    .
```

- 9) Construa um algoritmo para calcular a área de um retângulo cuja diagonal mede 15m e o perímetro mede 42m.
- 10) Você está pretendendo comprar um veículo e possui apenas R\$ 10.000,00 para efetuar o pagamento da entrada. Você quer pagar o veículo em 48 prestações. O vendedor lhe oferece um veículo no valor de R\$ 45.000,00 e lhe informa que a taxa de juros para veículos na faixa de preço entre 30 e 40 mil reais é de 1,78% ao mês. Construa um algoritmo para ler todos estes valores e calcular o valor que você pagará na prestação. Agora imagine a seguinte situação: Você já pagou 17 prestações deste veículo que comprou e pretende liquidar o financiamento. Liquidando o financiamento, você recebe um desconto dos juros que não irá mais pagar devido ao adiantamento das últimas parcelas (31 parcelas). Construa um algoritmo para calcular e mostrar o valor que você irá pagar ao liquidar o financiamento e mostrar o valor que você economizou ao liquidá-lo.

A.7. ENTENDER AS ESTRUTURAS DE SELEÇÃO (ALTERNATIVAS)

Exercícios da fase “Vejo, Escuto e Compreendo”

- 1) Faça um algoritmo para cadastrar as 4 notas bimestrais de 10 alunos em uma determinada matéria usando uma matriz 10 x 4. Ao final, mostrar a nota total de cada aluno aprovado naquela matéria, a nota média dos alunos aprovados e a nota média dos reprovados (Considere aprovado todo aluno cuja soma das notas for superior a 70 pontos).
- 2) Faça um algoritmo e seu respectivo fluxograma para ler 3 números inteiros em variáveis diferentes e mostrar o maior deles. Faça o mesmo algoritmo usando um vetor com 5 valores inteiros.
- 3) Faça um algoritmo e seu respectivo fluxograma para ler 5 números inteiros em variáveis diferentes e mostrá-los em ordem crescente. Faça o mesmo algoritmo usando um vetor com 4 valores inteiros.
- 4) Dê 3 exemplos de situações em que é necessário o uso de estruturas de seleção simples, composta, encadeada homogênea, encadeada heterogênea e de múltipla escolha.
- 5) Leia do teclado um vetor de 15 números inteiros. Em seguida, copie esses números para um segundo vetor de mesmo tamanho, alterando o sinal dos números que estiverem em posições pares. Ao final, mostre os valores do vetor resultante na tela.
- 6) Elabore um algoritmo que leia do teclado 20 números inteiros, armazene-os em uma matriz 4 x 5 e em seguida calcule e mostre a soma dos números pares digitados.
- 7) Elabore um algoritmo para calcular as raízes de uma equação do 2º grau. Lembre-se que não existe raiz quadrada de número negativo (pois não queremos trabalhar com números complexos).
- 8) Faça uma estrutura (identificada como “Atleta”) capaz de armazenar os dados (nome, altura, sexo e peso) de um indivíduo. Construa um algoritmo para ler os dados de 5 pessoas usando um vetor do tipo “Atleta”. Verifique e mostre os atletas que possuem um peso acima do ideal para a sua altura e sexo. Para isso, use as seguintes fórmulas:
 - Peso ideal para homens: $(72,7 * \text{Altura}) - 58$;
 - Peso ideal para mulheres: $(62,1 * \text{Altura}) - 44,7$.

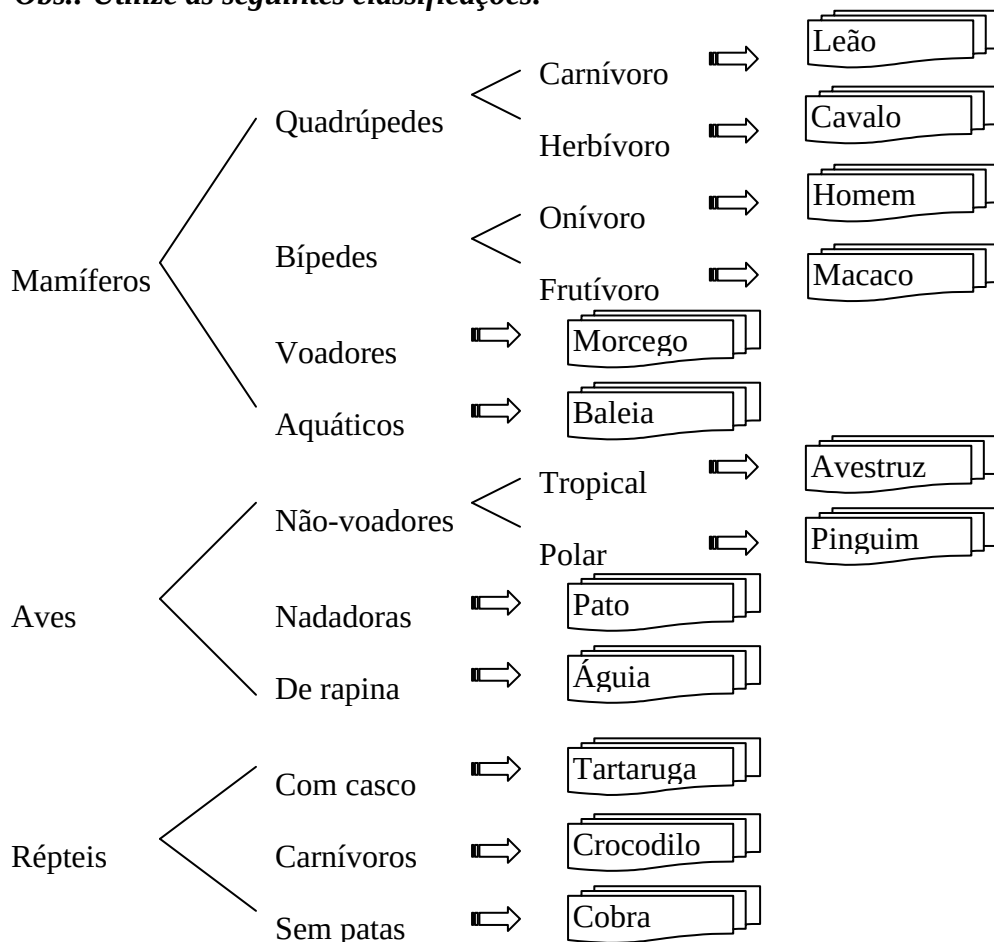
- 9) Elabore um algoritmo que leia o valor de 2 números inteiros e a operação aritmética desejada (em uma variável do tipo caractere); calcule, então, a resposta de acordo com a operação digitada. As operações aritméticas podem ser: (+, -, *, /, M, D, P). Onde **M** é **mod**, **D** é **div** e **P** é ****** ou potência.
- 10) Construa um algoritmo que seja capaz de concluir qual dentre os seguintes animais foi escolhido, através de perguntas e respostas. Animais possíveis: leão, cavalo, homem, macaco, morcego, baleia, avestruz, pinguim, pato, águia, tartaruga, crocodilo e cobra.

Exemplo 1:
 É mamífero? Sim.
 É quadrúpede? Sim.
 É carnívoro? Não.
 É herbívoro? Sim.
 Animal escolhido: cavalo.

Exemplo 2:
 É mamífero? Sim.
 É quadrúpede? Sim.
 É carnívoro? Não.
 É herbívoro? Não.
 Animal escolhido: não existe.

Exemplo 3:
 É mamífero? Não.
 É ave? Não.
 É réptil? Sim.
 Tem casco? Não.
 É carnívoro? Não.
 Sem patas? Sim.
 Animal escolhido: cobra.

Obs.: Utilize as seguintes classificações:



Exercícios da fase “Discuto, Aplico e Reflito”

- 1) Observe o seguinte enunciado: “Faça um algoritmo que leia um número e se ele for maior que 20, então escreva a metade desse número”.

```
Algoritmo “Cálculo da metade”
var
    NUMERO, METADE: real
inicio
    escreva(“INFORME O NÚMERO: ”)
    leia(NUMERO)
    se (NUMERO > 20) entao
        METADE = NUMERO / 2
    fimse
    escreva(“A METADE É: ”, METADE)
finalgoritmo
```

Analise a corretude do algoritmo proposto para resolvê-lo. O algoritmo está correto? Em caso negativo, justifique e reconstrua-o corretamente.

- 2) Observe o seguinte enunciado: “Faça um algoritmo que leia um número e se escreva se ele é par ou ímpar”.

```
Algoritmo “Par ou Ímpar”
var
    NUMERO: real
inicio
    escreva(“INFORME O NÚMERO: ”)
    leia(PAR)
    se (PAR mod 2 = 0) entao
        escreva(“O número digitado é par”)
    senão
        escreva(“O número digitado é ímpar”)
    fimse
finalgoritmo
```

Analise a corretude do algoritmo proposto para resolvê-lo. O algoritmo está correto? Em caso negativo, justifique e reconstrua-o corretamente.

- 3) Observe o seguinte enunciado: “Usando estrutura de seleção múltipla, escreva um algoritmo que leia o peso de um indivíduo na Terra e o número de um planeta e imprima o valor do seu peso neste planeta. A relação de planetas é dada a seguir juntamente com o valor das gravidades relativas à Terra”.

Número Planeta	Gravidade Relativa	Planeta
1	0,37	Mercúrio
2	0,88	Vênus
3	0,38	Marte
4	2,64	Júpiter
5	1,15	Saturno
6	1,17	Urano

Para calcular o peso no planeta use a fórmula:

$$P_{\text{planeta}} = \frac{P_{\text{terra}}}{10} * \text{gravidade}$$

Algoritmo “Cálculo do Peso Proporcional em Outros Planetas”

```

var
    op: inteiro;
    pterra: real;
inicio
    escreva("Escolha o planeta (1 – 6): ");
    leia(op);
    escreva("Informe seu peso: ");
    leia(pterra);
    escolha (op)
        caso 1:
            escreva("seu peso no planeta marcurio é: ", (pterra/10)*0.37);
        caso 2:
            escreva("seu peso no planeta vênus é: ", (pterra/10)*0.88);
        caso 3:
            escreva("seu peso no planeta marte é: ", (pterra/10)*0.38);
        caso 4:
            escreva("seu peso no planeta jupiter é: ", (pterra/10)*2.64);
        caso 5:
            escreva("seu peso no planeta saturno é: ", (pterra/10)*1.15);
        caso 6:
            escreva("seu peso no planeta marcurio é: ", (pterra/10)*1.17);
    fimsecolha
fimalgoritmo
  
```

Análise a corretude do algoritmo proposto para resolvê-lo. O algoritmo está correto? Em caso negativo, justifique e reconstrua-o corretamente.

- 4) Imagine um sistema de controle de Locadora de Fitas. Construa o tipo de dado “FILME” que possua os atributos que você julgar importantes para controlar empréstimo, cadastro por gênero (romance, ação, etc.) e categoria (lançamento, catálogo, etc.) do filme. Construa também um algoritmo para ler os dados de 5 filmes. Depois da leitura dos dados, o usuário deverá perceber na tela um menu com as seguintes opções:

1. Reajustar em 15% o valor de locação de todo filme de ação;
2. Reajustar em 17,5% o valor de locação de todo filme de romance;
3. Reajustar em 5% o valor de locação de todo filme da categoria lançamento.

Obs.: Imediatamente depois que o usuário escolher uma das opções do menu, o algoritmo deverá mostrar os títulos e os valores dos respectivos filmes.

- 5) Em uma fábrica trabalham homens e mulheres divididos em três classes: (1) Trabalhadores que fazem até 30 peças por mês – classe 1; (2) Trabalhadores que fazem de 31 a 35 peças por mês – classe 2; (3) Trabalhadores que fazem mais de 35 peças por mês – classe 3. A classe 1 recebe salário mínimo (R\$510,00). A classe 2 recebe salário mínimo mais 3% do salário mínimo por peça acima das 30 peças iniciais. A classe 3 recebe salário mínimo mais 5% do salário mínimo por peça fabricada acima das 30 peças iniciais. A fábrica possui 5 operários. Faça um algoritmo que leia, para cada operário: o seu número de matrícula (inteiro), o número de peças fabricadas no mês e seu sexo (1 para masculino ou 2 para feminino). Os dados devem ser armazenados em 3 vetores: vetNumOp, vetNumPecas e vetSexo respectivamente. O algoritmo deve calcular os salários dos funcionários, armazená-los em um quarto vetor (vetSalarios) e mostrar um relatório que contenha o número do operário, a quantidade de peças por ele fabricadas no mês e o seu salário. O algoritmo deve mostrar também o total da folha de pagamento da fábrica e o salário médio.
- 6) Faça um algoritmo para verificar se 3 valores digitados que armazenam os comprimentos das retas (A, B, C) podem formar um triângulo. Em caso positivo, mostre na tela o tipo do triângulo que eles podem formar (Equilátero, Isósceles ou Escaleno). Em caso negativo, informar ao usuário que eles não podem formar um triângulo e explicar o motivo.

- 7) Foi realizada uma pesquisa sobre algumas características físicas da população de uma certa região, a qual coletou os seguintes dados referentes a cada habitante para uma análise: (1) Sexo – M ou F; (2) Cor dos Olhos – Azuis, Verdes ou Castanhos; (3) Cor dos Cabelos – Louros, Castanhos ou Pretos; e (4) Idade. Construa uma estrutura de dados para armazenar estes dados. Faça um algoritmo para ler os dados de 10 indivíduos (de acordo com a estrutura criada) e mostre:
- A maior idade dos habitantes;
 - Quantos habitantes possuem os Olhos “Azuis”, são do sexo “F” e são maiores de idade;
 - Quantos habitantes possuem os Cabelos “Pretos”, são do sexo “M”, Olhos “Castanhos” e tem entre 18 e 35 anos de idade;
 - Quantos habitantes possuem cabelos “Louros” e são do sexo “F”.
- 8) Construa uma estrutura de dados e um algoritmo para calcule o imposto de renda de um grupo de 5 contribuintes, considerando que os dados de cada contribuinte (CPF, Número de Dependentes e Renda Mensal) são fornecidos pelo usuário. Para cada contribuinte será feito um desconto de 20% do salário mínimo (que deverá ser fornecido pelo usuário) por dependente em cima do valor base para cálculo do imposto de renda. Os valores da alíquota para cálculo do imposto são:
- Até 2 salários mínimos – Isento;
 - De 2 a 3 salários mínimos – 5%;
 - De 3 a 5 salários mínimos – 10%;
 - De 5 a 7 salários mínimos – 15%;
 - Acima de 7 salários mínimos – 20%;

Pergunta: O que aconteceria se o desconto do imposto por dependente fosse aplicado sobre o valor do imposto de renda já calculado? Explique.

- 9) Em uma eleição presidencial, existem 4 candidatos. Os votos são informados através de código (1, 2, 3 e 4 para os respectivos candidatos, 5 para os votos nulos e 6 para os votos em branco). Elabore um algoritmo que leia o voto de 10 eleitores e calcule e escreva:
- O total de votos apurados para cada candidato;
 - O total de votos nulos;
 - O total de votos em branco;

- O percentual dos votos brancos e nulos (somados) em relação ao total.

10) Dado o fragmento de algoritmo a seguir, responda: (1) Se A = verdade, B = verdade e C = falsidade, quais instruções serão executadas? (2) Se A = verdade, B = falsidade e C = verdade, quais instruções serão executadas? (3) Se A = verdade, B = verdade e C = verdade, quais instruções serão executadas? (4) Quais são os valores de A, B e C para que apenas a Instrução-6 seja executada?

```

Var
  A, B, C: lógico
Início
  Se (A) então Instrução-1
  Senão
    Se (B)
    Senão
      Se (C) então Instrução-2
      Senão
        Instrução-3
        Instrução-4
      Fimse
    Fimse
  Instrução-5
  Fimse
  Instrução-6
Fim

```

Exercícios da fase “Pratico, Faço e Crio”

- 1) Imagine que a velocidade máxima permitida em um cruzamento é de 60 km/h, vigiado por um semáforo equipado com radar e câmera de vigilância que fotografa todo veículo que cruza o sinal (considere uma via de um único sentido). Depois que o sinal fotografa a placa e registra a velocidade de uma quantia de 10 veículos, deve ser calculada a multa para todos os veículos infratores (aqueles que trafegavam acima dos 60 km/h). Para cada quilometro por hora acima da velocidade máxima permitida, deverá ser adicionado R\$ 5,00 à multa até o limite de 20% acima da velocidade permitida (Assim, se o condutor ultrapassar o semáforo a 71 km/h, deverá pagar uma multa de $11 * R\$ 5,00$). Para cada quilometro por hora acima de 20% da velocidade máxima permitida, deverá ser adicionado R\$ 10,00 à multa (Assim, se o condutor

ultrapassar o semáforo a 77 km/h, deverá pagar uma multa de $17 * R\$ 10,00$). Construa uma estrutura de dados para armazenar estas informações no cruzamento e o algoritmo para lê-los e calcular e mostrar o valor pago em multas e o valor médio pago em multas.

Obs.: Ao produzir o algoritmo, verifique se você usou a menor quantidade de linhas possível e se as seleções estão otimizadas.

2) Um cinema possui capacidade de 10 lugares e está sempre lotado. Certo dia, cada espectador respondeu a um questionário, no qual constava: (1) sua idade; (2) sua opinião em relação ao filme segundo as seguintes notas: (A) – Ótimo; (B) Bom; (C) Regular; (D) Ruim; e (E) Péssimo. Construa uma estrutura de dados capaz de armazenar os dados de resposta do questionário. Elabore um algoritmo para ler a resposta de todos os espectadores e calcule e imprima:

- A quantidade de respostas “Ótimo”;
- A diferença percentual entre respostas “Bom” e “Regular”;
- A média de idade das pessoas que responderam “Ruim”;
- A porcentagem de respostas “Péssimo” e a idade da pessoa mais velha que escolheu esta opção;
- A diferença de idade entre a pessoa mais nova que respondeu “Ótimo” e a pessoa mais nova que respondeu “Ruim”.

Obs.: Ao produzir o algoritmo, verifique se você usou a menor quantidade de linhas possível e se as seleções estão otimizadas.

3) Uma empresa fez uma pesquisa para saber se as pessoas gostaram ou não de um de seus novos produtos lançado no mercado. Para isso coletou: o sexo do entrevistado (“M” ou “F”), sua idade e sua resposta (“S” ou “N”). Sabendo-se que foram entrevistadas 10 pessoas, elabore um algoritmo para calcular e informar:

- Número de pessoas que responderam “Sim”;
- Número de pessoas que responderam “Não”;
- Quantas pessoas maiores de 18 anos gostaram do produto;
- Quantas pessoas menores de 18 anos não gostaram do produto;
- Quantas pessoas maiores de 18 anos, do sexo feminino, não gostaram do produto;
- Quantas pessoas menores de 18 anos, do sexo masculino, gostaram do produto.

Obs.: Ao produzir o algoritmo, verifique se você usou a menor quantidade de linhas possível e se as seleções estão otimizadas.

- 4) O código a seguir resolve o seguinte enunciado: “Faça um algoritmo para ler 5 valores e mostrar o maior deles”. O algoritmo está otimizado? Ele pode ser diminuído em questão de quantidade de linhas de código e pode executar mais rapidamente? Justifique. Se a resposta for “sim”, reescreva-o de modo a executar mais rapidamente.

```
Algoritmo “Mostra o maior de 5 números”
Var
    A, B, C, D, E: inteiro
Inicio
    // Seção de leitura dos valores de A, B, C, D e E
    .
    .
    Se (A >= B e A >= C e A >= D e A >= E)
        Escreva(“O maior número é :”, A)
    Fimse
    Se (B >= A e B >= C e B >= D e B >= E)
        Escreva(“O maior número é :”, B)
    Fimse
    Se (C >= A e C >= B e C >= D e C >= E)
        Escreva(“O maior número é :”, C)
    Fimse
    Se (D >= A e D >= B e D >= C e D >= E)
        Escreva(“O maior número é :”, D)
    Fimse
    Se (E >= A e E >= B e E >= C e E >= D)
        Escreva(“O maior número é :”, E)
    Fimse
Fimalgoritmo
```

- 5) Em geral, otimização de código é muito difícil. Na maioria dos casos, o programador é obrigado a sacrificar recursos de armazenamento (espaço em disco ou memória) em detrimento de performance (eficiência) e vice-versa. Dê exemplo de pelo menos um caso como este que você já tenha presenciado ao desenvolver um algoritmo. Explique por que isso ocorre.

- 6) Na maioria das linguagens de programação, a instrução vista na caixa (1) pode ser mais eficiente que a combinação das instruções vista na caixa (2) a seguir. Por que e em que caso isto ocorre?

```
(1)  se (X>B e B>A) então
      ...
      ...
      fimse
```

```
(2)  se (X>B) então
      se (B>A) então
      ...
      ...
      fimse
      fimse
```

- 7) O código a seguir resolve o seguinte enunciado: “Faça um algoritmo para ler 3 valores e mostrá-los em ordem decrescente”. O algoritmo está otimizado? Ele pode ser diminuído em questão de quantidade de linhas de código e pode executar mais rapidamente? Justifique. Se a resposta for “sim”, reescreva-o de modo a executar mais rapidamente.

```
Algoritmo “Mostra 3 números em ordem decrescente”
Var
    A, B, C: inteiro
Inicio
    // Seção de leitura dos valores de A, B e C
    .
    .
    Se (A>=B e B>=C)
        Escreva(“A ordem dos números é :”, A, B, C)
    Fimse
    Se (A>=C e C>=B)
        Escreva(“A ordem dos números é :”, A, C, B)
    Fimse
    Se (B>=A e A>=C)
        Escreva(“A ordem dos números é :”, B, A, C)
    Fimse
    Se (B>=C e C>=A)
        Escreva(“A ordem dos números é :”, B, C, A)
    Fimse
    Se (C>=A e A>=B)
        Escreva(“A ordem dos números é :”, C, A, B)
    Fimse
    Se (C>=B e B>=A)
        Escreva(“A ordem dos números é :”, C, B, A)
    Fimse
Fimalgoritmo
```

8) Qual é mais eficiente? A seleção realizada na caixa (1) ou a realizada na caixa (2)?

Justifique.

```
(1)  escolha (op)
      caso 1:
          ...
          ...
      caso 2:
          ...
          ...
      fimsecolha
```

```
(2)  se (op = 1) então
      ...
      ...
      fimse
      se (op = 2) então
          ...
          ...
      fimse
```

9) A instrução vista na caixa (1) é equivalente à combinação das instruções vista na caixa (2) a seguir? Justifique.

```
(1)  se (X>B ou B>A) então
      ...
      ...
      fimse
```

```
(2)  se (X>B) então
      ...
      ...
      fimse
      se (B>A) então
          ...
          ...
      fimse
```

10) A instrução vista na caixa (1) é mais eficiente que a combinação das instruções vista na caixa (2) a seguir? Por que? Qual delas pode ser considerada otimizada? Só existe um critério pelo qual ela poderia ser considerada otimizada. Qual é esse critério?

```
(1)  se (X>B ou B>A) então
      ...
      ...
      fimse
```

```
(3)  se (X>B) então
      ...
      ...
      aborte
      fimse
      se (B>A) então
          ...
          ...
      fimse
```

A.8. ENTENDER AS ESTRUTURAS DE REPETIÇÃO (LAÇOS)

Exercícios da fase “Vejo, Escuto e Compreendo”

- 1) Faça um algoritmo para ler, calcular e mostrar a soma e a média aritmética de 50 números inteiros digitados pelo usuário. Resolva o problema de três maneiras distintas (Com teste programado, com teste no início e com teste no final).
- 2) Faça um algoritmo para calcular a área de um círculo, um triângulo, um quadrado e um retângulo N vezes, ou seja, quantas vezes o usuário assim desejar. Elabore um algoritmo que contenha o seguinte menu: (1) – Calcular Área do Triângulo; (2) – Calcular Área do Retângulo; (3) – Calcular Área do Círculo; (4) – Calcular Área do Quadrado; e (5) – Sair. Resolva o problema de três maneiras distintas (Com teste programado, com teste no início e com teste no final).
- 3) Faça um algoritmo para carregar uma matriz 40 x 40 e escreva o maior elemento da diagonal superior e a soma dos elementos da diagonal inferior usando laço com teste programado.
- 4) Execute um chinesinho ou tabela de testes para o algoritmo abaixo e escreva o valor final de cada uma das variáveis.

Algoritmo “Chinesinho”

Var

A, B, C, D, E: inteiro

Início

B $\leftarrow$ 3

Para A de 1 ate 200 faca

B $\leftarrow$ B + 9

C $\leftarrow$ B div A

Fimpara

D $\leftarrow$ 0

Enquanto D < 2000 faca

E $\leftarrow$ D * 2

D $\leftarrow$ D + 2

Fimenquanto

Fimalgoritmo

- 5) Faça um algoritmo para cadastrar os dados de 100 funcionários de uma empresa com a seguinte estrutura: Nome, endereço, cidade, telefone, salário, cargo, departamento. Ao final: (1) mostrar todos os funcionários do departamento Pessoal; (2) mostrar todos os funcionários de uma cidade escolhida pelo usuário; e (3) mostrar o valor que a empresa gasta por mês com folha de pagamento, levando-se em consideração somente o salário de cada funcionário. Use repetição com teste programado.
- 6) Faça um algoritmo para cadastrar os dados de **N** funcionários (até o máximo de 500 funcionários) de uma empresa com a seguinte estrutura: Nome, endereço, cidade, telefone, salário, cargo, departamento. Elabore um algoritmo que contenha o seguinte menu: (1) - mostrar todos os funcionários do departamento Pessoal; (2) - mostrar todos os funcionários de uma cidade escolhida pelo usuário; (3) - mostrar os 10 primeiros funcionários cadastrados; (4) mostrar o valor que a empresa gasta por mês com folha de pagamento; e (5) Sair. Resolva o problema usando repetição com teste no final.
- 7) Faça um algoritmo que carregue 5 vetores de 5 posições do tipo inteiro e armazene-os em uma matriz 5 X 5. Cada vetor deverá ser uma linha da matriz. Calcule e mostre o determinante desta matriz.
- 8) Faça um algoritmo para calcular e mostrar a soma dos 10 primeiros elementos da seguinte série:

$\frac{1}{1}$	+	$\frac{1}{2}$	+	$\frac{1}{4}$	+	$\frac{1}{8}$	+	$\frac{1}{16}$	+	$\frac{1}{32}$	...
---------------	---	---------------	---	---------------	---	---------------	---	----------------	---	----------------	-----

- 9) Faça um algoritmo para calcular a soma dos **N** primeiros elementos da seguinte série (Como você não sabe o valor de **N**, pergunte ao usuário quantos elementos ele deseja ter na série):

$\frac{2}{3}$	+	$\frac{4}{6}$	+	$\frac{8}{9}$	+	$\frac{16}{12}$	+	$\frac{32}{15}$	+	$\frac{64}{18}$	...
---------------	---	---------------	---	---------------	---	-----------------	---	-----------------	---	-----------------	-----

- 10) Faça um algoritmo para calcular mostrar os **N** primeiros elementos da série Fibonacci. Como você não sabe o valor de **N**, pergunte ao usuário quantos elementos ele deseja ter na série. Veja exemplo a seguir:

1	1	2	3	5	8	13	21	34	55	...
---	---	---	---	---	---	----	----	----	----	-----

Exercícios da fase “Discuto, Aplico e Reflito”

- 1) Faça um algoritmo para ler uma matriz 100 x 10 de caracteres (usando laço com teste programado) que se refere a respostas de 10 questões de múltipla escolha (os valores permitidos são: “a”, “b”, “c”, e “d”) de um máximo de 100 alunos (as respostas de cada aluno estão representadas em uma linha). Leia também um vetor de 10 posições de caracteres (usando laço com teste no início) contendo o gabarito de respostas (obviamente, cujos valores devem ser representados por “a”, “b”, “c”, ou “d”). Seu algoritmo deverá comparar as respostas de cada candidato com o gabarito e produzir um vetor de inteiros como resultado (com a quantidade de acertos para cada aluno correspondente uma linha do vetor de respostas”).
- 2) Um cinema possui capacidade de 1000 lugares (Perceba que desta vez a quantidade de entrevistados é tão grande que torna necessário o uso de instruções de repetição e um menu de escolhas) e está sempre lotado. Certo dia, cada espectador respondeu a um questionário, no qual constava: (1) sua idade; (2) sua opinião em relação ao filme segundo as seguintes notas: (A) – Ótimo; (B) Bom; (C) Regular; (D) Ruim; e (E) Péssimo. Construa uma estrutura de dados capaz de armazenar os dados de resposta do questionário. Elabore um algoritmo que contenha o seguinte menu:
 - Responder o questionário;
 - Mostrar a quantidade de respostas “Ótimo”;
 - Mostrar a diferença percentual entre respostas “Bom” e “Regular”;
 - Mostrar a média de idade das pessoas que responderam “Ruim”;
 - Mostrar a porcentagem de respostas “Péssimo” e a idade da pessoa mais velha que escolheu esta opção;
 - Mostrar a diferença de idade entre a pessoa mais nova que respondeu “Ótimo” e a pessoa mais nova que respondeu “Ruim”.
 - Sair

- 3) Faça um algoritmo para ler um vetor que contém o valor da **distância** em quilômetros entre cidades (Cadastre 100 cidades). O vetor abaixo mostra o exemplo:

Posição	1	2	3	4	5	6	7	8	Valores fictícios. Não leia estes valores.
Cidade	1-2	2-3	3-4	4-5	5-6	6-7	7-8	8-9	
Dist.	103	47	38	234	456	17	290	132	

Ou seja, se o vetor contivesse os valores acima, a distância entre a cidade **1** e a cidade **2** seria 103 km, a distância entre a cidade **2** e a cidade **3** seria 47 km, a distância entre a cidade **3** e a cidade **4** seria 38 km, e assim por diante.

Depois que as distâncias entre as cidades forem digitadas, faça um menu com as seguintes opções:

1 – Calcular a distância entre duas cidades. (Ao escolher esta opção, o usuário deverá digitar quais as duas cidades (origem e destino) para as quais deseja calcular a distância. Se o usuário digitar cidade origem = 2 e cidade destino = 7, então você deve calcular a distância entre a cidade 2 e a cidade 7. Neste caso, o resultado seria $47 + 38 + 234 + 456 + 17 = 792$. Finalizado este processo, mostre a distância entre as duas cidades).

2 – Calcular a quantidade de gasolina em litros para percorrer a distância entre uma cidade origem e uma cidade destino. (Ao escolher esta opção, o usuário deverá digitar quais as duas cidades (origem e destino) para as quais deseja calcular a distância. Depois que o algoritmo calcular esta distância, o usuário deverá digitar a autonomia do veículo, ou seja, qual a quantidade de quilômetros que o veículo faz com um litro de combustível. Depois disso, você deve calcular e mostrar a quantidade de combustível (em litros) que o veículo deverá consumir para percorrer a distância entre as duas cidades).

- 4) Faça um algoritmo para ler cinco vetores que contém a **distância** em quilômetros entre uma cidade **origem** e outra **destino**. Cada um dos vetores possui uma possibilidade de caminho entre origem e destino. Os vetores abaixo demonstram o exemplo:

Posição	1	2	3	4
Cidade	Origem – C2	C2 – C3	C3 – C4	C4 – Destino
Caminho1	103	47	38	234
Cidade	Origem – C3	C3 – C4	C4 – C2	C2 – Destino
Caminho2	201	38	12	58
Cidade	Origem – C4	C4 – C3	C3 – C2	C2 – Destino
Caminho3	170	38	47	58
Cidade	Origem – C3	C3 – C2	C2 – C4	C4 – Destino
Caminho4	201	47	12	234

Valores fictícios.
Não leia estes
valores.

Depois que as distâncias entre as cidades forem digitadas (para cada um dos quatro caminhos possíveis), faça um menu com as seguintes opções:

- 1 – Mostrar qual é o menor caminho entre as duas cidades
- 2 – Mostrar a distância percorrida entre as duas cidades através do menor caminho
- 3 – Mostrar a distância percorrida entre as duas cidades através do maior caminho
- 4 – Mostrar a distância média percorrida se considerar os 4 caminhos
- 5 – Mostrar a menor distância computada entre duas cidades
- 6 – Sair

Obs.: A leitura das distâncias entre as cidades (dos vetores de distâncias) ocorre apenas uma vez para cada vetor. O menu deve ficar na tela várias vezes até que o usuário escolha a opção 6.

- 5) Em uma fábrica trabalham homens e mulheres divididos em três classes: (1) Trabalhadores que fazem até 30 peças por mês – classe 1; (2) Trabalhadores que fazem de 31 a 35 peças por mês – classe 2; (3) Trabalhadores que fazem mais de 35 peças por mês – classe 3. A classe 1 recebe salário mínimo (R\$510,00). A classe 2 recebe salário mínimo mais 3% do salário mínimo por peça acima das 30 peças iniciais. A classe 3 recebe salário mínimo mais 5% do salário mínimo por peça fabricada acima das 30 peças iniciais. A fábrica possui N operários (até um máximo de 500 operários). O algoritmo será usado para calcular a folha de pagamento da empresa e deverá armazenar, para cada operário, o seu número de matrícula (inteiro), o número de peças fabricadas no mês, seu sexo (“1” para masculino ou “2” para feminino) e o

- salário (calculado pelo seu algoritmo seguindo as classes anteriormente citadas). Os dados devem ser armazenados em um único vetor com esta estrutura. Faça um menu com as seguintes opções: (1) – Cadastrar funcionário na folha; (2) - Mostrar relatório de folha de pagamento que contenha o número do operário, a quantidade de peças por ele fabricadas no mês e o seu salário; (3) – Mostrar valor total da folha de pagamento da fábrica; (4) – Mostrar salário médio pago no mês; (5) – Mostrar a quantidade de funcionários cadastrados até o momento; e (5) Sair.
- 6) Escreva um algoritmo para ler e armazenar, para vários clientes (com limite máximo de 50 clientes), as seguintes informações: código do cliente (inteiro), nome (máximo de 40 caracteres), idade, salário, produto_última_compra (deve ser representado pelo código do produto, portanto, deve ser inteiro). As informações devem ser armazenadas em um vetor com esta estrutura. Construa o seguinte menu de opções: (1) – Cadastrar novo cliente; (2) – Mostrar todos os clientes cadastrados; (3) – Consultar um cliente especificado pelo usuário; (4) – Mostrar o código do produto mais comprado ultimamente; e (5) – Sair.
- 7) Identifique e descreva o que o algoritmo a seguir está fazendo. Que valor conterà o vetor X1 quando o algoritmo finalizar?

```
Algoritmo "AAAAAAA"
Var
    Const cont = 50
    X1: vetor [1..cont] de caracteres
Inicio
    X1 = "Renata é uma menina levada da Brega"
    Para (int i=0;i<cont;i++) faça
        Para (int j=i+1;j<cont;j++) faça
            Se (X1[i] > X1[j]) então
                aux=X1[i];
                X1[i]=X1[j];
                X1[j]=aux;
        Fimse
    Fimpara
Fimpara
Fimalgoritmo
```

- 8) Analise o algoritmo abaixo e responda: (1) O que representam os elementos do vetor Y?; (2) O que define o valor final de K?; (3) Mostre os valores armazenados no vetor X depois que o algoritmo finalizar;

Algoritmo “GGGGGGGG”

Var

I, J, N, K: inteiro

X, Y: vetor [1..20] de inteiro

Início

N = 20

I = 1

Enquanto (I <= N) faça

 X[I] = 1

 I = I + 1

Fimenquanto

I = 2

Enquanto I <= N faça

 J = 2

 Enquanto ((I * J) <= N) faça

 X[I * J] = 0

 J = J + 1

 Fimenquanto

 I = I + 1

 Enquanto (X[I] = 0 e I <= N) faça

 I = I + 1

 Fimenquanto

Fimenquanto

I = 1

K = 1

Enquanto (I <= N) faça

 Se (X[I] = 1) então

 Y[K] = I

 K = K + 1

 Fimse

 I = I + 1

Fimenquanto

K = K - 1

Fimalgoritmo

9) Considerando o algoritmo a seguir, indique o valor armazenado em:

- vet3[1];
- vet2[5];
- vet3[7];
- vet1[8];
- vet3[19].

```
início
| declare Vet1 [20], Vet2 [20], Vet3 [20], i : inteiro
| para i de 0 até 19 passo 1 faça
|   | Vet1[i] ← i
|   | Vet2[i] ← 20 - i
| fim para
| para i de 0 até 19 passo 1 faça
|   | Vet3[i] ← Vet1[i] + Vet2[i]
| fim para
fim
```

10) Considerando o algoritmo a seguir, indique o valor armazenado em:

- Mat1[1][0];
- Mat1[3][2];
- Mat1[4][0];
- “j” e “i” quando a execução do algoritmo terminar.

```
início
| declare Mat1[4][3], i, j : inteiro
| i ← 0
| enquanto (i < 4) faça
|   | j ← 0
|   | enquanto (j < 3) faça
|   |   | Mat1[i][j] ← (i * 4) - 3
|   |   | j ← j + 1
|   | fim enquanto
|   | i ← i + 1
| fim enquanto
| i ← 0
| enquanto (i < 4) faça
|   | j ← 0
|   | enquanto (j < 3) faça
|   |   | escreva Mat1[i][j]
|   |   | j ← j + 1
|   | fim enquanto
|   | i ← i + 1
| fim enquanto
fim
```

Exercícios da fase “Prático, Faço e Crio”

- 1) Faça um algoritmo para ler 30 números em um vetor de inteiros, ordená-los e mostrá-los em ordem crescente. Em seguida, mostre-os também em ordem decrescente. Resolva o problema usando laço com teste programado, em seguida, resolva usando laço com teste no início e, por fim, resolva usando laço com teste no final. O método de ordenação utilizado deverá ser o **ShellSort**.
- 2) Faça um algoritmo para ler os dados de 1000 clientes em uma estrutura que contém nome, endereço e telefone. Mostre os nomes dos clientes em ordem alfabética. Use o método de ordenação conhecido como **Inserção**.
- 3) Escreva um algoritmo para ler e armazenar, para vários clientes (com limite máximo de 50 clientes), as seguintes informações: código do cliente (inteiro), nome (máximo de 40 caracteres), idade, salário, produto_última_compra (deve ser representado pelo código do produto, portanto, deve ser inteiro). As informações devem ser armazenadas em um vetor com esta estrutura. Construa o seguinte menu de opções: (1) – Cadastrar novo cliente; (2) – Mostrar todos os clientes cadastrados; (3) – Mostrar todos os clientes cadastrados em ordem alfabética; (4) – Mostrar todos os clientes cadastrados em ordem de idade; e (5) – Sair.

*Obs.: A opção 3 e a opção 4 do menu devem gerar dois novos vetores para guardar os clientes dispostos em ordem alfabética e em ordem de idade respectivamente. O método de ordenação utilizado devem ser o **HeapSort**.*

- 4) Faça um algoritmo para calcular o **mmc** entre dois números.
- 5) Faça um algoritmo para calcular o **mdc** entre dois números.
- 6) O fragmento de algoritmo a seguir não é eficiente. Identifique o que ele se propõe a fazer e, baseado nisto, reconstrua-o para torná-lo mais eficiente possível.

```
J = 1
Para (I = 0; I <= 2000; I = I + 1) faça
    Se ((I mod 3) = 0)
        Escreva (I)
        J = J * I
    Fimse
Fimpara
Escreva (J)
```

- 7) O algoritmo a seguir lê 100 elementos de um vetor e mostra estes elementos na ordem invertida à qual foram carregados. Contudo, existem erros lógicos (ou semânticos) nos laços. Identifique-os e construa o fluxograma correspondente corrigindo os erros do algoritmo.

Algoritmo “Lógica incorreta”

Var

A, B, C: inteiro

D, E: vetor [1..20] de inteiro

Inicio

Para A de 1 ate 200 faca

Escreva(“Digite um valor para armazenar no vetor”)

Leia(D[A])

Fimpara

B = 0

Enquanto B < 20 faca

E[B] = D[B] * 3

B = B + 1

Fimenquanto

Fimalgoritmo

- 8) O fragmento de algoritmo a seguir calcula e mostra o quadrado de 100 números lidos. O algoritmo pode ser otimizado em relação ao tempo de processamento. Reconstrua-o para torná-lo mais eficiente possível. Identifique e descreva o critério que você está sacrificando ao tornar o algoritmo mais eficiente.

Para (I = 1; I <= 100; I = I + 1) faça

Escreva(“Digite um número”)

Leia(VET[I])

Fimpara

Para (I = 1; I <= 100; I = I + 1) faça

VET[I] = VET[I] ** 2

Fimpara

Para (I = 1; I <= 100; I = I + 1) faça

Escreva(“O quadrado do ”, I, “º número é: ”, VET[I])

Fimpara

- 9) Os algoritmos dos exemplos 1 e 2 a seguir resolvem o problema de ler todos os elementos de um vetor de 100 inteiros. Porém, embora o laço do exemplo 1 execute menos vezes que o laço do exemplo 2 (economizando tempo de processamento), a construção de códigos desta maneira é amplamente desaconselhada, não só na academia, mas também na indústria de software. Analise e descreva o motivo disso.

```
1) Para (I = 1; I <= 100; I = I + 1) faça
    Escreva("Digite um número")
    Leia(VET[I])
    I = I + 1
    Escreva("Digite um número")
    Leia(VET[I])
    I = I + 1
    Escreva("Digite um número")
    Leia(VET[I])
    I = I + 1
    Escreva("Digite um número")
    Leia(VET[I])
Fimpara
```

```
2) Para (I = 1; I <= 100; I = I + 1) faça
    Escreva("Digite um número")
    Leia(VET[I])
Fimpara
```

- 10) Os algoritmos dos exemplos 1 e 2 a seguir resolvem o problema de ler todos os elementos de uma matriz 8 x 8. Qual dos dois exemplos é mais eficiente em tempo de processamento? Justifique. Qual você aconselha seguir? Justifique.

```
1) Para (I = 1; I <= 8; I = I + 1) faça
    Para (J = 1; J <= 8; J = J + 1) faça
        Escreva("Digite um número")
        Leia(VET[I][J])
    Fimpara
Fimpara
```

```
2) J = 1;
I = 0;
Para (K = 1; K <= 64; K = K + 1) faça
    I = I + 1
    Escreva("Digite um número")
    Leia(VET[I][J])
    Se ((K mod 8) = 0)
        I = 0
        J = J + 1
    Fimse
Fimpara
```

A.9 ENTENDER MODULARIZAÇÃO E CONSTRUIR SOFTWARE MODULAR

Exercícios da fase “Vejo, Escuto e Compreendo”

- 1) Faça um algoritmo para calcular e mostrar a soma dos **N** primeiros elementos da seguinte série usando função. A função deverá receber o valor de **N** como parâmetro de entrada (que deverá ser previamente digitado pelo usuário) e deverá retornar o valor resultante calculado.

$$\frac{1}{1} + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} \dots$$

- 2) Faça um algoritmo para calcular a soma dos **N** primeiros elementos da seguinte série usando função. A função não receberá valor como parâmetro de entrada (o valor de **N** deverá ser digitado pelo usuário e sua implementação deverá ocorrer dentro da função) e deverá retornar o valor resultante calculado.

$$\frac{2}{3} + \frac{4}{6} + \frac{8}{9} + \frac{16}{12} + \frac{32}{15} + \frac{64}{18} \dots$$

- 3) Faça um algoritmo para calcular e mostrar os **N** primeiros elementos da série Fibonacci usando procedimento. O procedimento deverá receber o valor de **N** como parâmetro de entrada (que deverá ser previamente digitado pelo usuário) e deverá mostrar o valor resultante calculado.

$$1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad 13 \quad 21 \quad 34 \quad 55 \quad \dots$$

- 4) Faça uma função para ler e retornar os dados de um Certificado de Registro de um Veículo (CRV). A função não deverá receber nenhum parâmetro de entrada, pois os dados serão lidos dentro da própria função. A estrutura é a que se segue: ModeloVeiculo, AnoFabricação, AnoModelo, Placa, Chassi, Renavan.
- 5) Transforme o algoritmo já pronto para o enunciado “Faça um algoritmo para ler 30 números em um vetor de inteiros, ordená-los e mostrá-los em ordem crescente. Em

seguida, mostre-os também em ordem decrescente. Resolva o problema usando laço com teste programado. O método de ordenação utilizado deverá ser o **ShellSort** em um algoritmo modularizado. Utilize uma função para ordenar o vetor, um procedimento para mostrar o vetor em ordem crescente e um terceiro procedimento para mostrá-lo em ordem decrescente.

- 6) Transforme o algoritmo já pronto para o enunciado “Faça um algoritmo para calcular o **mmc** entre dois números” em um algoritmo modularizado. Utilize uma função que receba os dois números e retorne o valor do **mmc** entre eles.
- 7) Transforme o algoritmo já pronto para o enunciado “Faça um algoritmo para calcular o **mdc** entre dois números” em um algoritmo modularizado. Utilize um procedimento que leia os dois números (não deve receber nenhum parâmetro de entrada) e mostre o valor do **mmc** entre eles.
- 8) Usando os dois algoritmos anteriores (“cálculo do mmc” e “cálculo do mdc” desta mesma lista), exatamente da maneira como foram elaborados, construa um algoritmo com as seguintes opções de menu:

Que opção deseja executar?

- 1 – Calcular mmc**
- 2 – Calcular mdc**
- 3 - Sair**

- 9) Faça um algoritmo modularizado para calcular e mostrar a média aritmética entre 100 números lidos do teclado. Construa um procedimento para armazenar os valores em um vetor de inteiros, uma função que recebe o vetor carregado e calcula e retorna a soma dos seus elementos e uma função que recebe o valor da soma dos elementos e a quantidade de elementos e calcula e retorna a média aritmética entre eles.
- 10) Escreva um algoritmo modularizado para ler e armazenar, para vários clientes (com limite máximo de 50 clientes), as seguintes informações: código do cliente (inteiro), nome (máximo de 40 caracteres), idade, salário, produto_última_compra (deve ser representado pelo código do produto, portanto, deve ser inteiro). As informações devem ser armazenadas em um vetor com esta estrutura. Construa o seguinte menu de opções: (1) – Cadastrar novo cliente; (2) – Mostrar todos os clientes cadastrados; (3) – Consultar um cliente especificado pelo usuário; (4) – Mostrar o código do produto mais comprado ultimamente; e (5) – Sair.

Obs.: Cada opção do menu (exceto a opção Sair) deve ser implementada através de um procedimento independente. Os procedimentos deverão ser chamados dentro do menu de opções.

Exercícios da fase “Discuto, Aplico e Reflito”

- 1) Construa um algoritmo modularizado para calcular e mostrar os **N** primeiros números primos.
- 2) Construa uma função recursiva para calcular o fatorial de um número qualquer.
- 3) Construa um procedimento recursivo para calcular e mostrar os **N** primeiros elementos da série Fibonacci.
- 4) Analise, identifique e descreva o propósito da função a seguir:

```
inteiro CALC(N1, N2: inteiro)
var
    I, Result: inteiro
inicio
    Result = 1
    Para (I = 1; I <= N2; I = I + 1)
        Result = Result * N1
    Fimpara
    Retorne Result
Fim
```

- 5) A função abaixo calcula o fatorial de um número qualquer usando laço com teste no início. Seu código está correto? Se “não”, justifique e construa uma função para retornar o fatorial de um número corretamente. Faça uma tabela de teste para calcular o fatorial de 5.

```
inteiro fatorial(n: inteiro )
var
    I: inteiro
    FAT: real
Inicio
    I = 1
    FAT = 1
    enquanto (I <= n) faça
        FAT = FAT * I
        I = I + 1
    fimenquanto
    retorne FAT
fim
```

- 6) Faça uma função que receba como parâmetro de entrada um número de CPF (conjunto de 11 caracteres) e calcule se o CPF é válido. A função deve retornar o valor “verdade”, no caso de CPF válido e o valor “falso”, no caso de CPF inválido.
- 7) Faça uma função que receba como parâmetro de entrada um número de CNPJ (conjunto de 14 caracteres) e calcule se o CNPJ é válido. A função deve retornar o valor “verdade”, no caso de CNPJ válido e o valor “falso”, no caso de CNPJ inválido.
- 8) Faça um algoritmo modularizado para controlar as operações de compra e venda de uma loja. O algoritmo deve conter 4 estruturas diferentes. A primeira estrutura é utilizada para controlar o cadastro de “Clientes” da loja (Nome, CPF, Endereço). A segunda estrutura é utilizada para controlar o cadastro de “Fornecedores” da loja (RazãoSocial, CNPJ, Telefone). A terceira estrutura é utilizada para armazenar os dados dos produtos comprados pela loja para venda (Código, Descrição, ValorVenda, Quantidade). A quarta estrutura é utilizada para armazenar as transações de compra e venda da loja (Código, Data, Quantidade, ValorCusto). Toda vez que a loja resolver fazer uma compra, o usuário do sistema deverá fazer uma busca pelo fornecedor. Se o fornecedor não existir no vetor de fornecedores, o mesmo deverá ser cadastrado. Depois disso, o usuário deverá fazer uma busca pelo produto e, novamente, se o produto não existir no vetor de produtos, o mesmo deverá ser cadastrado. Por fim, a transação é armazenada no vetor de transações. Toda vez que a loja resolver fazer uma venda, o usuário do sistema deverá fazer uma busca pelo cliente. Se o cliente não existir no vetor de clientes, o mesmo deverá ser cadastrado. Depois disso, o usuário deverá fazer uma busca pelo produto e, novamente, se o produto não existir no vetor de produtos, deverá ser enviada uma mensagem de falha ao usuário. Se o produto existir, a transação é armazenada no vetor de transações. Use função e procedimento onde você considerar necessário. Use as funções de validação de CPF e CNPJ quando estiver cadastrando clientes e fornecedores. O menu de opções é o que se segue:

Que opção deseja executar?

- 1 – Cadastrar Clientes**
- 2 – Cadastrar Fornecedores**
- 3 – Cadastrar Produtos**
- 4 – Comprar Produtos**
- 5 – Vender Produtos**
- 6 – Buscar Produtos**
- 7 – Buscar Fornecedor**
- 8 – Buscar Cliente**
- 9 - Sair**

- 9) Escreva uma função chamada DATA_VALIDA que receba como parâmetros de entrada uma data (DIA, MÊS e ANO) e retorne “verdadeiro”, se a data for válida ou “falso”, se a data for inválida (considerando os anos bissextos).
- 10) Escrever um algoritmo que leia dois vetores com 10 elementos inteiros cada. Considerando cada vetor como sendo um conjunto, construa uma função para gerar um terceiro vetor que seja a união dos dois primeiros. Os elementos no final do vetor solução que não contiverem valores válidos deverão conter o valor -32768.

Obs.: Não devem haver elementos repetidos no vetor solução.

Exercícios da fase “Prático, Faço e Crio”

- 1) Identifique e descreva o propósito da função a seguir. Ela pode ser otimizada? Em caso positivo, reescreva-a mais otimizada possível.

```
inteiro CALC(N1, N2: inteiro)
var
    I, Result: inteiro
inicio
    Result = 1
    Para (I = 1; I <= N2; I = I + 1)
        Result = Result * N1
    Fimpara
    Retorne Result
Fim
```

- 2) A função a seguir pode ser otimizada? Em caso positivo, reescreva-a mais otimizada possível.

```
inteiro fatorial(n: inteiro )
var
    I: inteiro
    FAT: real
Inicio
    I = 1
    FAT = 1
    enquanto (I <= n) faça
        FAT = FAT * I
        I = I + 1
    fimenquanto
    retorne FAT
fim
```

- 3) Existem algumas maneiras de se calcular o π . Pesquise a respeito disso e construa uma função para calculá-lo.
- 4) Faça uma pesquisa a respeito dos conceitos de Coesão e Acoplamento. Elabore um resumo de 4 páginas. Dê 2 exemplos de funções e/ou procedimentos cujo código não é considerado Coeso. Explique porque o código apresenta baixa coesão nos dois exemplos e reescreva-os de modo que se tornem Coesos. Dê 2 exemplos de funções e/ou procedimentos cujo código é considerado muito Acoplado. Explique porque o código apresenta alto Acoplamento nos dois exemplos e reescreva-os de modo que se tornem menos Acoplados.
- 5) Escreva um algoritmo que leia e armazene uma sequência de 100 caracteres em um vetor chamado TEXTO. Depois, deve ser lida e armazenada uma subsequência de 5 caracteres em um vetor chamado PADRÃO. Desenvolva uma função que receba como parâmetro de entrada o vetor TEXTO e o vetor PADRÃO para verificar se a subsequência PADRÃO aparece completa e na mesma ordem em algum ponto do vetor TEXTO, caso ocorra, retornar primeira posição do vetor TEXTO onde a subsequência ocorre PADRÃO. Se não ocorrer, retorne o valor "0".

Exemplo vetor TEXTO:

R	E	N	A	T	A		É		U	M	A		M	E	N	I	N	A	.	.	.	.
---	---	---	---	---	---	--	---	--	---	---	---	--	---	---	---	---	---	---	---	---	---	---

Exemplo vetor PADRÃO:

É		U	M	A
---	--	---	---	---

Exemplo de resposta ao usar a função: **8**

- 6) Analisando o código de algoritmo a seguir, pode ser considerado que seu acoplamento está alto (seu bom funcionamento depende de outros fatores externos a ele). Identifique e descreva o(s) motivo(s) e reescreva-o de maneira a tornar o seu acoplamento controlado.

Algoritmo “SSSSSS”

```
// Definição do Procedimento CALC( )
CALC( )
var
    I: inteiro
inicio
    Para (I = 1; I <= 20; I = I + 1)
        Escreva(“Digite um número”)
        Leia(N1[I])
    Fimpara
Fim

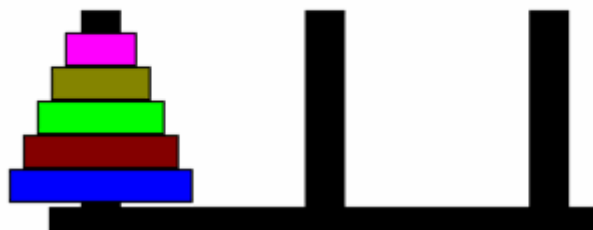
// Início do Algoritmo Principal
Var
    N1: vetor [1..20] de inteiro
Inicio
    .
    .
    .
Fimalgoritmo
```

- 7) Construa um algoritmo para controle de uma vídeolocadora. Deverão existir estruturas para armazenar dados de clientes, filmes e locação. Todas as funcionalidades do algoritmo deverão estar modularizadas, altamente coesas e pouco acopladas. As funcionalidades necessárias para o desenvolvimento de um sistema de locadora deverão ser analisadas e identificadas por você (Faça uma entrevista com um funcionário de alguma vídeolocadora). Contudo, elabore pelo menos 10 funcionalidades.

- 8) Analisando o código de algoritmo a seguir, pode ser considerado que sua coesão está baixa (possui procedimentos ou funções que realizam mais de uma atividade). Identifique e descreva o(s) motivo(s) e reescreva-o de maneira a torná-lo mais coeso. Caso seja necessário, construa outros procedimentos.

```
CALC1(N1, N2: vetor [1..20] de inteiro)
var
    I, J: inteiro
inicio
    Para (I = 1; I <= 20; I = I + 1)
        Escreva("Digite um número")
        Leia(N1[I])
    Fimpara
    J = 1
    Para (I = 1; I <= 20; I = I + 1)
        Se ((N1[I] mod 2) = 0)
            N2[J] = N1[I] ** 2
            J = J + 1
        Fimse
    Fimpara
    Para (I = 1; I <= J; I = I + 1)
        Escreva("O novo número é: ", N2[I])
    Fimpara
Fim
```

- 9) Um jogo designado por “Torres de Hanói” popularizou-se na Europa no final século passado. Este jogo consistia em três cilindros verticais nos quais encaixava-se um conjunto de 64 discos de tamanhos diferentes. De acordo com a figura (exemplo para 5 discos) o objetivo do jogo é o de transportar os discos do cilindro da esquerda para o cilindro da direita, com o auxílio do cilindro do meio. Os discos devem ser movidos um de cada vez e nunca deverá ficar um disco maior colocado sobre um disco menor. Este é um dos clássicos problemas que só podem ser solucionados recursivamente. Escreva uma função (use 3 vetores para simular os cilindros) que resolva este problema.



- 10) Analisando o código de algoritmo a seguir, pode ser considerado que seu acoplamento está alto (seu bom funcionamento depende de outros fatores externos a ele) e sua coesão está baixa (possui procedimentos ou funções que realizam mais de uma atividade). Identifique e descreva o(s) motivo(s) e reescreva-o de maneira a tornar o seu acoplamento controlado e aumentar a sua coesão. Caso seja necessário, construa outros procedimentos.

```
Algoritmo "TTTTTTTTT"
  CALC2()
  var
    I, Soma: inteiro
    Media: real
  inicio
    Para (I = 1; I <= 20; I = I + 1)
      Escreva("Digite um número")
      Leia(N1[I])
    Fimpara
    Soma = 0
    Para (I = 1; I <= 20; I = I + 1)
      Soma = Soma + N1[I]
    Fimpara
    Media = Soma / 20
    Escreva("A média dos números digitados é: ", Media)
  Fim
// Início do Algoritmo Principal
Var
  N1: vetor [1..20] de inteiro
Início
  .
  .
Fimalgoritmo
```

APÊNDICE B

EXERCÍCIOS DE LÓGICA

Este Apêndice se encontra dividido em exercícios para treinar a atenção e exercícios para treinar o raciocínio lógico.

B.1 EXERCÍCIOS PARA TREINAR A ATENÇÃO

- 1) Você está participando de uma corrida de “F1” e de repente ultrapassa o segundo colocado. Em que lugar você fica?
- 2) Você está dirigindo um ônibus para Salvador. Em uma parada descem 25 passageiros e seguem 20, no próximo ponto sobem 7, no terceiro ponto sobem 8 e descem 12. No ponto final um passageiro não quer descer. Qual o nome do motorista?
- 3) Quando você perde alguma coisa, por que sempre a encontra no último lugar que a procura?
- 4) Em um relógio com ponteiros, os ponteiros de Hora e Minuto se encontram após 1:00 h, após 2:00 h, após 3:00 h, ... Você saberia calcular com a maior precisão possível as horas, minutos em que tais “encontros” ocorrem? Você saberia mostrar qual e que tipo de Matemática é usada para resolver este problema?
- 5) Marcelo e Jáder resolvem fazer uma aposta no jogo de dados. Utilizando apenas dois dados, estipulam que o resultado de um dado deve ser multiplicado pelo resultado do outro. Se este resultado for um número par, o Marcelo vence, se o resultado for um número ímpar, quem vence é o Jáder. Qual dos dois tem mais chances de vencer?

- 6) Dois pais e dois filhos saíram para caçar patos. Cada um deles acertou em um pato e nenhum atirou no pato que outro acertou. Entretanto, somente três (3) patos foram abatidos. Qual a explicação lógica para este fato?
- 7) Um grande empresário, na necessidade de ir a São Paulo, chegou a seu guarda noturno e ordenou que ele o acordasse às 6 horas da manhã em ponto. Exatamente às 6:00 da manhã o guarda acordou o empresário e disse:
- Patrão, estou com um mal pressentimento: sonhei esta noite que o senhor teria um acidente com o avião. Me permita sugerir que não viaje.
- O empresário não deu ouvidos ao guarda. Sem incidentes, chegou a São Paulo e por telefone mandou demiti-lo. Por quê?
- 8) Três gatos comem três ratos em três minutos. Cem gatos comem cem ratos em quantos minutos?
- 9) Descubra o erro !! (sofisma algébrico)
- Vou provar que 2 é igual a três !! Partiremos da igualdade: $2-2 = 3-3$
A diferença $(2-2)$ pode ser escrita sob a forma de produto: $2(1-1)$.
Da mesma forma $(3-3) = 3(1-1)$.
Ora, então poderemos escrever:
 $2(1-1) = 3(1-1)$
Cancelando-se em ambos os membros dessa igualdade o fator comum $(1-1)$
Resulta que $2 = 3$
Onde está o erro ????
- 10) Se um bezerro pesa 75 kg mais meio bezerro, quanto pesa um bezerro inteiro?
- 11) Você está dando voltas em uma praça. Existem duas pessoas à sua frente e duas pessoas atrás de você, mas, no total (com você) são 3. Como?
- 12) Os curiosos olhavam uma pessoa que havia sido assassinada e o médico de imediato percebeu que tinha sido morte por estrangulamento, mas ficou quieto ouvindo os comentários que certamente levaria ao criminoso:
- Eu acho que a Marli deveria confessar o crime e sofrer as consequências.
 - Um momento, respondeu a Marli, o Alexandre e o Valdir eram os únicos interessados na morte dessa pessoa.
- O Valdir disse:
- Para mim foi o André, o Renato ou o Bruno. Bem, talvez tenha sido o Guilherme, o Cláudio, o Marcelo, a Marlene, a Vera, a Antonia Fatobene. Chiiii, é tanta gente que nem sei quem poderia ter estrangulado esta pessoa.
- Sem saber o que fazer, a Vilma e a Sheyla choravam.
- É um absurdo, disseram Felipe, Estevão, Jemima e Darinho, temos que descobrir o criminoso e colocá-lo na prisão.

Quem é o assassino?

- 13) Três amigos foram à um bar e cada um tomou cinco cervejas. A conta, que ficou em 30 reais, foi dividida entre eles, de forma que cada um pagou 10 reais. O dono do bar acabou dando um desconto e cobrou apenas 25 reais, devolvendo os outros 5. Os três amigos deram ao garçom 2 reais de gorjeta, e cada um deles ficou com um 1 real de troco. Problema: Cada um dos amigos acabou pagando pela conta 9 reais (10 reais menos 1 real de troco), que multiplicado por 3 é igual a 27 reais, mais os 2 reais dados de gorjeta ao garçom é igual a 29 reais. Onde foi parar o real que falta?

B.2 EXERCÍCIOS PARA TREINAR O RACIOCÍNIO LÓGICO

- 14) Três algarismos são representados pelas letras A, B e C, na soma abaixo. Quais são esses algarismos?

$$\begin{array}{r} A A A \\ B B B \\ + C C C \\ \hline A B B C \end{array}$$

- 15) Um homem, que pesa 100 quilos, e seus 2 filhos, um pesando 40 quilos e o outro pesando 60, precisam atravessar um rio. O único barco disponível só pode carregar até 100 quilos de cada vez. Como eles poderão chegar à outra margem?
- 16) Ontem à noite, dona Viviane, mãe do querido professor Fábio, perguntou-lhe quantos brigadeiros deveriam ser feitos para comemorar seu aniversário. A resposta foi a seguinte “Devem ser feitos 240 brigadeiros, pois convidei n pessoas e gostaria que cada convidado recebesse o mesmo número de doces”. Se não vierem 10 convidados, quantos brigadeiros a mais receberá cada pessoa? A resposta estava na ponta da língua “Cada pessoa receberá 4 brigadeiros a mais”. Quantas pessoas serão convidadas?
- 17) Um elevador pode levar, ou 20 adultos ou 24 crianças. Se 15 adultos já estão no elevador, quantas crianças podem entrar?
- 18) Cinco marinheiros se colocam lado a lado para receber as ordens do comandante do navio. Tente nomeá-los, da esquerda para a direita, de acordo com as informações:
- Anderson está entre Jorge e Cláudio;

- Humberto está à esquerda de Cláudio;
- Jorge não está ao lado de Humberto;
- Humberto não está ao lado de Rafael.

Atenção! A sua esquerda não é a esquerda dos marinheiros (por causa do ponto de vista).

- 19) Tenho o quádruplo da idade que você tem. Daqui a 4 anos terei o triplo da sua idade. Quais são as nossas idades?
- 20) Como se pode repartir para três pessoas, 21 tonéis de vinho, se 7 tonéis estão vazios, 7 tonéis estão cheios e 7 tonéis estão pela metade, de modo que no final da divisão cada pessoa tenha a mesma quantidade de vinho e de tonéis.
- 21) Como se pode repartir, igualmente para duas pessoas, 8 litros de vinho que estão em uma vasilha maior, sabendo-se que estas pessoas possuem somente duas vasilhas vazias – uma com capacidade para 5 litros e outra com capacidade para 3 litros.
- 22) Como se pode repartir igualmente para duas pessoas, 16 litros de vinho que estão em uma vasilha maior, sabendo-se que as pessoas possuem somente duas vasilhas vazias, uma com capacidade para 11 litros e outra com capacidade para 6 litros.
- 23) Carlos possui 4 baldes sem graduação com capacidades de 18, 10, 8 e 2 litros. O balde maior está cheio de água, e os outros estão vazios. Carlos precisa repartir essa água em três porções iguais usando apenas os 4 baldes com, no máximo 6 movimentos. Como isso pode ser feito?
- 24) Há 10 litros de vinho em 3 vasilhas, com capacidades iguais a 6 litros, 3 litros e 7 litros respectivamente. A primeira contém 4 litros de vinho, a segunda está vazia e a terceira contém 6 litros de vinho. Como se pode repartir em duas partes iguais o vinho, usando apenas estas três vasilhas?
- 25) Pretende-se construir uma coleção de “pesos” que possam ser utilizados para medir massas de objetos com uma balança contendo dois pratos equilibrados. De que forma uma barra metálica com massa de 40 Kg poderá ser cortada em apenas 4 partes de modo a se poder pesar objetos desde 1 Kg até 40 Kg.
- 26) Um velho tinha três filhos e lhes deu a ordem que depois de morto, deveriam dividir os 35 camelos que possuía, de modo que o primeiro filho deveria receber a metade deles, o segundo deveria receber um terço e ao último caberia um nono. Como não houve concordância entre eles, foram até um sábio que também possuía um camelo. Como foi que o sábio realizou a divisão de forma que todos os filhos ficaram satisfeitos com a divisão e no final até mesmo o sábio acabou ganhando algo?

27) Com três 5 e as operações aritméticas elementares (+, -, *, /), obtenha:

- O número 0;
- O número 2;
- O número 4;
- O número 5;
- Outros números.

28) Diz-se que um número é perfeito quando ele é igual à soma de todos os seus divisores, exceto ele próprio como, por exemplo, $1+2+4+7+14 = 28$. Qual o menor número perfeito?

29) Considere um jogo de dominó e as suas 28 peças.

- Qual é a soma dos pontos de todas as peças?
- Qual é a soma dos pontos de todas as peças de forma que um dos lados é zero?
- Qual é a soma dos pontos de todas as peças de forma que um dos lados tem um número par?
- Qual é a soma dos pontos de todas as peças de forma que ambos os lados têm números pares?
- Qual é a soma dos pontos de todas as peças de forma que um dos lados tem um número ímpar?
- Qual é a soma dos pontos de todas as peças de forma que ambos os lados têm números ímpares?

30) Você está numa cela onde existem duas portas, cada uma vigiada por um guarda. Existe uma porta que dá para a liberdade, e outra para a morte. Você está livre para escolher a porta que quiser e por ela sair. Para isso, você poderá fazer apenas uma pergunta a um dos dois guardas que vigiam as portas. Um dos guardas sempre fala a verdade, e o outro sempre mente. Você não sabe quem é o mentiroso e quem fala a verdade. Que pergunta você faria?

31) Você é prisioneiro de uma tribo indígena que conhece todos os segredos do Universo e portanto sabem de tudo. Você está para receber sua sentença de morte. O cacique o desafia: “Faça uma afirmação qualquer. Se o que você falar for mentira você morrerá na fogueira, se falar uma verdade, você será afogado. Se não pudermos definir sua afirmação como verdade ou mentira, nós te libertaremos”. O que você diria?

32) Um joalheiro tinha 9 pérolas. Apesar de serem idênticas, ele sabia que uma delas era falsa e que esta era um pouco mais leve do que as outras. Como a diferença não podia

ser constatada a olho nu, ele usou uma balança de precisão e, com apenas 2 pesagens, pode separar a pérola diferente. Como ele fez isso?

- 33) Existem em uma floresta, um lobo e uma raposa. Toda segunda, terça e quarta, o lobo só dizia mentiras (nada do que ele dizia nestes dias era verdade) e, no restante dos dias, ele só dizia verdades. Enquanto que, a raposa, dizia somente mentiras em toda quinta, sexta e sábado (nada do que ela falava nestes dias era verdade) e, no restante dos dias, só falava verdades. No domingo ambos falavam verdade. Um dia eles se encontraram na floresta e o lobo disse:

- Olá, dona Raposa. Ontem eu menti.

E a raposa respondeu:

- Oi, seu Lobo. Ontem eu também menti.

Qual foi o dia da semana em que houve esta conversa?

- 34) O Sr. X chegou ao escritório que lhe prestava serviços, onde trabalhavam, Paulo, Marcos, Daniel, Roberto e Carlos que eram, o chefe, o contador 2 auxiliares e o Office-boy:

- Gostaria de falar com o Chefe.

Marcos respondeu:

- Ele saiu com o Paulo.

- Será que ele vai demorar?

- Não sei. Ele disse alguma coisa, Daniel?

Quem respondeu foi o office-boy que já estava saindo:

- Ele vai demorar, sim. Pediu-me para avisar o Carlos

O contador falou:

- Talvez possamos atendê-lo, O Marcos pode levantar sua ficha ...

Não ! Eu quero falar é com o chefe mesmo!

- Então, sentimos muito ...

O Sr. X não resolveu o seu problema, e você, resolveu este ?

Qual o nome do chefe, do contador, do Office-boy e dos 2 auxiliares?

- 35) Nas olimpíadas, Fred, Fabricio e Ricardo participaram das mesmas três provas. Cada um conseguiu um primeiro lugar, um segundo e um terceiro. Descubra qual a colocação de deles em cada prova, sabendo que: 1) Fabricio venceu Ricardo no salto com vara; 2) Frederico chegou atrás de Fabricio na corrida; 3) Ricardo não ficou em primeiro lugar no arremesso de disco.

- 36) Em uma transportadora de grãos, foram acondicionados os produtos a serem transportados naquela tarde em três caixas. Na primeira caixa, foram colocados dois sacos de ervilha. Na segunda caixa, colocaram dois sacos de soja. Na terceira caixa, colocaram um saco de ervilha e um saco de soja. Por descuido, as caixas foram

etiquetadas erroneamente. Assim, nenhuma caixa continha o que estava descrito na etiqueta. Na caixa “A”, a etiqueta dizia que seu conteúdo era de dois sacos de soja. Na caixa “B”, a etiqueta informava que ali se encontravam dois sacos de ervilha. A etiqueta da caixa “C” afirmava que seu conteúdo era de um saco de ervilha e outro de soja (caixa mista). Como descobrir o conteúdo correto de cada caixa, abrindo somente uma delas e retirando apenas um saco?

- 37) Ademar comprou uma bicicleta e resolveu doá-la de presente a um de seus três filhos. Para ser imparcial, ele estipulou que, aquele que conseguisse acertar a cor da bicicleta, seria premiado e a levaria de presente. Com o intuito de tornar a brincadeira mais justa, ele informou que a cor da bicicleta, ou era preta, ou vermelha ou azul. O filho mais velho disse que achava que a bicicleta não era preta. O filho do meio disse que, para ele, a bicicleta ou era vermelha ou azul. Já o caçula, achava que a cor da bicicleta era vermelha. Ademar disse que pelo menos um deles havia acertado, e que pelo menos um havia errado. Qual a cor da bicicleta?

APÊNDICE C

COMO AGIR DURANTE O EMPARELHAMENTO DE PROGRAMADORES?

Programação em Duplas (*Pair Programming* – PP) é ideal para alunos “ativos” que aprendem por interação social e profissionais que possuem habilidades para o trabalho colaborativo e facilidades em comunicação. Quando ambos, aprendizes e profissionais, não estão acostumados com práticas colaborativas ou não concordam com o protocolo de programação emparelhada, é necessário um período inicial de aculturação com o objetivo de eliminar possíveis focos de resistência – participantes iniciantes de emparelhamentos costumam se mostrar demasiadamente resistentes a esta nova técnica de programação, pois são retirados de sua zona de conforto, exigindo mudanças profundas em seus comportamentos.

O protocolo de PP mais utilizado na literatura envolve três regras básicas: (1) dois desenvolvedores de software compartilham um único monitor e teclado – exercendo papéis de piloto e navegador; (2) os sócios trocam seus papéis regularmente, normalmente em um intervalo típico de 20 minutos; (3) o rodízio dos pares é incentivado para promover a transmissão de informação e conhecimento entre os membros de cada equipe.

Embora o protocolo direcione para o funcionamento do processo de emparelhamento, as diretrizes comportamentais dos atores participantes do processo de desenvolvimento de software (Instrutores ou Gerentes de Projeto e Alunos ou Desenvolvedores – Pilotos e Navegadores) encontradas a seguir norteiam atitudes que promovem seu sucesso.

É importante destacar que este guia foi desenvolvido com o objetivo de nortear o caminho de pesquisadores que pretenderem aplicar a técnica de Programação em Duplas em disciplinas de “Programação de Computadores”.

No decorrer das seções seguintes, quando não houver a explícita intenção de destacar a sua natureza, os atores participantes de projetos emparelhados (Aprendizes ou Desenvolvedores) serão referenciados como sócios. Nos demais casos, será usada a própria nomenclatura atribuída ao ator (Instrutores, Gerentes de Projeto, Aprendizes e/ou Desenvolvedores).

C.1 DIRETRIZ 1 – DETALHAMENTO DOS PASSOS E REGRAS DE PP

É fundamental que os atores descritos anteriormente não tenham dúvidas sobre como agir para diminuir o risco de enganos e atitudes prejudiciais ao processo de PP. Adicionalmente, quando os sócios são treinados em habilidades colaborativas, eles se tornam mais receptivos e menos resistentes à prática, isso ocorre porque muitos nunca tiveram experiências precedentes em equipes.

C.1.1 Instrutores ou gerentes de projeto devem

- Descrever e detalhar todos os passos do processo de emparelhamento, todos os objetivos e todas as regras de funcionamento;
- Treinar os sócios em habilidades colaborativas – membros emparelhados precisam: (1) saber se comunicar com seus sócios; (2) colaborar – produzir junto; (3) ter compromisso com as atividades e com o sócio; (4) seguir éticas de trabalho emparelhado; (5) ser adaptáveis às mudanças;
- Considerar as seções iniciais normalmente como seções de ajustes – é especialmente importante ser paciente neste momento e ajudar os sócios durante a adaptação;
- Mostrar aos sócios a importância da troca de conhecimentos e habilidades entre eles;

- Monitorar a efetividade dos emparelhamentos e intervir quando necessário para ensinar habilidades colaborativas;
- Conscientizar-se que a transição do processo de trabalho individual para o processo emparelhado tira os sócios de sua zona de conforto, o que poderia explicar uma certa resistência inicial;
- Permitir o trabalho individual quando a dupla estiver resolvendo tarefas triviais dentro da atividade proposta. Mesmo assim, atividades realizadas individualmente devem passar pelo crivo do sócio que se encontrava ausente no momento em que foram realizadas;
- Periodicamente, enfatizar o que é esperado dos pares durante os emparelhamentos (comportamentos colaborativos e os papéis interdependentes do piloto e do navegador);
- Em ambientes acadêmicos, avaliar as atividades desenvolvidas emparelhadamente – os alunos parecem ser comprometidos em atividades quando eles sabem que estão sendo avaliados;
- Ainda em ambientes acadêmicos, deixar claro aos alunos que as avaliações rotineiras continuarão sendo de natureza individual, pois, ao ingressarem no mercado de trabalho, eles não serão alocados somente em atividades de programação emparelhada; o conhecimento individual também será exigido.

C.1.2 Estudantes ou desenvolvedores devem

- Manter-se ativos durante o desenvolvimento, evitando permanecer em uma posição passiva quando estiverem exercendo o papel de navegador;
- Prestar atenção no seu sócio, inclusive, direcionar os olhos para o sócio quando estiver falando e escutá-lo cuidadosamente;
- Quando agindo como pilotos, descrever ao navegador seus ideais ao codificar linhas vitais ou estruturas importantes e não triviais;
- Quando agindo como pilotos e nos casos em que alguma atividade for realizada individualmente, discutir com o navegador, outrora ausente, sobre todas as suas ações antes que a atividade seja considerada finalizada;
- Manter respeito pelo sócio – navegadores não controlam o mouse ou o teclado, pilotos não tomam posse de materiais de referência utilizados pelos navegadores. Não obstante, a

discordância é natural, e deve ser solucionada respeitosamente;

- Procurar não se ocupar de tarefas dispersas durante o trabalho como, por exemplo, leitura de e-mails ou navegação na *web* quando seus sócios esperam contribuição contínua.

C.2 DIRETRIZ 2 – COMUNICAÇÃO ATIVA ENTRE NAVEGADOR E PILOTO

É importante incentivar a comunicação entre os sócios de modo que o navegador consiga acompanhar o raciocínio do piloto. Um baixo índice de comunicação entre eles, certamente, acarretará em diminuição ou perda dos benefícios advindos do emparelhamento.

C.2.1 Instrutores ou gerentes de projeto devem

- Incentivar o piloto para que, em atos de codificação, explique detalhadamente e discuta com o navegador que resultados pretende obter com cada instrução ou conjunto de instruções adicionadas ao código;
- Incentivar o navegador a inquirir o piloto sempre que se sentir confuso ou que não concordar com seu sócio;
- Incentivar a comunicação ativa entre piloto e navegador;
- Compreender que nem todos os diálogos entre os participantes de emparelhamentos devem estar voltados à execução de determinada tarefa – às vezes, uma piada ou um momento de descontração são ideais para aumentar a confiança entre os sócios, desde que não comprometa a continuidade do trabalho.

C.2.2 Estudantes ou desenvolvedores devem

- Enquanto pilotos, comunicar suas atitudes aos navegadores e esclarecer seus objetivos

com cada instrução ou conjunto de instruções;

- Enquanto navegadores, solicitar explicações dos pilotos em relação a partes confusas do programa em codificação;
- Procurar compreender os benefícios da comunicação e utilizá-la com o objetivo de aprimorar suas habilidades.

C.3 DIRETRIZ 3 – CULTURA ORIENTADA PARA A PP E PARA ATIVIDADES COLABORATIVAS

O acultramento dos sócios em atividades colaborativas é primordial para que a implantação de PP seja bem sucedida – A cultura da dominância-complacência deve ser combatida, se possível, extinta. Discussões de como a implementação de programação emparelhada se difere desses ideais deve ser encorajada. É importante ressaltar que as pessoas não estão acostumadas a colaborar, em vez disso, são altamente competitivas, motivo pelo qual a PP poderia falhar. O comportamento colaborador é contrastado com um comportamento competidor e individual.

Em PP, é esperado que os sócios compartilhem seus conhecimentos e suas habilidades com seus sócios, assim como devem absorver os conhecimentos e habilidades deles. Entretanto, algumas pessoas são mais socializadas que outras. Para essas pessoas, trabalhar colaborativamente é definitivamente benéfico, para as demais, é necessário construir continuamente a cultura da colaboração.

C.3.1 Instrutores ou gerentes de projeto devem

- Incentivar que os sócios estimulem e participem da obtenção de conhecimentos e habilidades de seus sócios;
- Promover discussões sobre temas polêmicos relacionados à programação nas salas de aula no intuito de incentivar o conflito e a produção de consenso;

- Promover dinâmicas de grupo;
- Desestimular a competição dentro das duplas. A competição “controlada” entre duplas dentro de uma classe pode ser aceitável, porém, é necessário um estudo aprofundado de quais seriam os benefícios advindos de tal prática;
- Criar um método oficial que não permite a um membro da dupla finalizar o programa individualmente sem a devida revisão do seu par, porém, é importante compreender que forçar um sócio a depender de um sócio incerto pode criar insatisfação ou um conflito entre honestidade e complacência;
- Nos casos de dominância-complacência, semanalmente, discutir com os sócios alguns dos ideais da programação emparelhada – sócios não devem impor suas opiniões (dominância) e não devem simplesmente acatar opiniões (complacência) antes de discuti-las;
- Incentivar a discussão aberta de assuntos como propriedade coletiva de código, metas de manutenção e expectativas da indústria com o objetivo de ajudar na criação de uma cultura para a programação emparelhada;

C.3.2 Estudantes ou desenvolvedores devem

- Compreender que seus sócios podem ter idéias ou visões diferentes, e que isto é benéfico ao processo de desenvolvimento de software – você pode dizer, “Oh, isso é uma idéia boa, me deixe pensar nisso!”. Isso o torna mais flexível e ajuda a compreender novas visões sobre o assunto ou problema;
- Compreender que, no mundo real, não existirá escolha quando seu chefe impuser o trabalho emparelhado com um sócio específico;
- Procurar verificar se compreendem sobre o que constitui um bom comportamento da equipe e se conhecem bem as regras de *Pair Programming*;
- Ser ativos e ter voluntariedade positiva, participando e ajudando quando for necessário;
- Demonstrar vontade de aprimorar-se, de aprender cada vez mais;
- Mostrar-se dispostos a seguir o protocolo de *Pair Programming*;
- Ter uma mente aberta e desejar ouvir e aprender mais sobre o que seu sócio agrega ao trabalho;

- Ser bons ouvintes;
- Saber como admitir confusão – Focalize nas instruções e seja específico. “Eu não sei onde você clicou para abrir aquela caixa de diálogo”, é uma declaração que pode conduzir a eliminar a confusão de um sócio. Por outro lado, “Eu não entendo nada disto”, provavelmente não ajudará a eliminar confusão e pode tornar o relacionamento entre os sócios negativamente conflituoso;
- Saber questionar seu sócio – Por exemplo, um membro da dupla pode sentir que não estão sendo executadas instruções corretamente. Neste caso, ao argumentar, “eu entendo o que você fez no passo *X*, mas o que foi feito no passo *Y* não parece seguir as regras propostas durante o projeto!” pode ser útil ao entendimento do processo. Porém, declarar “companheiro (a), você não sabe o que está fazendo!” provavelmente não será útil e pode construir um sentimento de resistência entre os sócios;
- Compreender que os benefícios da colaboração superam em muito os benefícios da atividade individual;
- Procurar desvencilhar-se da necessidade de serem recompensados pelo êxito em suas atividades individuais;
- Fazer autocritica em relação aos seus atos como piloto ou navegador, procurando corrigir atitudes incondizentes a cada papel.

C.4 DIRETRIZ 4 – INCENTIVO AO RELATO DE CASOS DE INCOMPATIBILIDADE

Duplas incompatíveis são aquelas em que os participantes não conseguem sintonizar seus objetivos, motivações, ou personalidades. Resumindo, duplas que não conseguem trabalhar harmoniosamente são consideradas incompatíveis e, conseqüentemente, degradam o processo de colaboração.

C.4.1 Instrutores ou gerentes de projeto devem

- Incentivar que os sócios relatem casos de incompatibilidade da dupla;
- Conscientizar os sócios sobre a inexistência de penalidades quando da incompatibilidade das duplas.

C.4.2 Estudantes ou desenvolvedores devem

- Compreender que a incompatibilidade degrada o processo de emparelhamento e que ambos os sócios estão sendo prejudicados;
- Procurar desenvolver senso autocrítico e verificar se o motivo da incompatibilidade da dupla é ou não ocasionado pelas suas atitudes.

C.5 DIRETRIZ 5 – HONESTIDADE, COLABORAÇÃO E SENSO DE RESPONSABILIDADE

Sócios, na maioria das vezes “alunos”, motivados pela natureza corporativista de seus atos, tendem a proteger os colegas que não participam idealmente dos emparelhamentos. Esta atitude deve ser desencorajada. Por outro lado, não é aceitável falar ou pensar coisas como, “Você causou um erro em nosso projeto” ou “Você é o responsável por aquele defeito”. Ao invés, são incentivadas expressões como, “Nós falhamos no projeto!” ou melhor ainda, “Nós falhamos ao procurar os defeitos!”.

C.5.1 Instrutores ou gerentes de projeto devem

- Incentivar a honestidade, a colaboração e o senso de responsabilidade;
- Desencorajar a submissão de tarefas com o nome de um sócio que não tiver contribuído ao processo;
- Quando em ambientes acadêmicos, promover avaliações dos estudantes somente com informações positivas – Os alunos não se sentem confortáveis quando são obrigados a avaliar a participação dos colegas. Por esse motivo, quando for o caso, os formulários para avaliações de pares podem ser compostos somente por informações positivas, como, por exemplo, “Favor identificar uma contribuição que tenha sido especialmente válida dada pelo seu parceiro”. O uso desta prática pode dar informações úteis de como a turma desenvolve um espírito colaborativo;
- Desencorajar atitudes corporativistas – os sócios não devem proteger parceiros descomprometidos;
- Conscientizar os membros das duplas de que são responsáveis pela participação e contribuição do seu parceiro;
- Conscientizar os sócios de que, em programação emparelhada, os dois programadores se tornam um. Não deve haver nenhuma competição entre os dois; ambos têm que trabalhar para um propósito singular, como se o software fosse produzido por uma mente singular.

C.5.2 Estudantes ou desenvolvedores devem

- Compreender que é mais importante ser honesto do que seguir fielmente as normas do protocolo de programação emparelhada;
- Compreender que o corporativismo pode estar contribuindo para o insucesso do seu parceiro em relação ao aprendizado e às avaliações individuais futuras;
- Conscientizar-se da importância da contribuição de seu parceiro – os membros da dupla devem compartilhar seus conhecimentos e qualificações. A cada membro, é necessário buscar novos conhecimentos de outros que os tenham adquirido;
- Tomar a iniciativa para reduzir a velocidade de um sócio avançado que estiver conduzindo

a atividade em uma velocidade incompatível;

- Compreender que a propriedade e a responsabilidade dos projetos desenvolvidos são atribuídas igualmente aos dois sócios – o par precisa confiar no julgamento do seu parceiro e ser leal a ele sem, necessariamente, se omitir quando ele precisar ser advertido;
- Criar oportunidades para promover aprendizado em seu parceiro;
- Criar oportunidades para a negociação de aspectos chave do projeto, do código e dos testes.

C.6 DIRETRIZ 6 – COMPATIBILIDADE DAS DUPLAS

Parece intuitivo que o emparelhamento funciona bem quando existe compatibilidade entre os pares. A incompatibilidade pode tornar os sócios menos receptivos à prática de *Pair Programming*. Duas abordagens podem aumentar as chances de compatibilidade da dupla: (1) cada sócio produz uma lista com os prováveis parceiros ou com as qualidades que gostariam que eles tivessem; e (2) rodízio dos pares pode ser incentivado nas seções iniciais de emparelhamentos até que seus participantes encontrem o par mais compatível a eles.

C.6.1 Instrutores ou gerentes de projeto devem

- Incentivar a composição de duplas compatíveis – a compatibilidade pode ocorrer por semelhança de objetivos, por habilidade, por personalidade, por afinidade, por motivação ou por outros aspectos;
- Experimentar composições de grupos diferentes para poucas semanas no começo do semestre ou do projeto no intuito de prevenir algumas das mais óbvias dificuldades dos sócios, como por exemplo, uma dupla na qual ninguém deseja tomar iniciativa, ou uma dupla na qual todos os membros são tímidos.

C.6.2 Estudantes ou desenvolvedores devem

- Procurar verificar que tipo de compatibilidade (por semelhança de objetivos, por habilidade, por personalidade, por afinidade, por motivação ou por outros aspectos) é mais benéfico a eles;
- Conscientizar-se de que a compatibilidade por afinidade potencializa o risco do corporativismo, aumentando significativamente a possibilidade de falha no emparelhamento (diminuição ou extinção dos benefícios de *Pair Programming*).

C.7 DIRETRIZ 7 – TRATAMENTO ESPECIAL AOS RESISTENTES

O uso de atividades colaborativas pode soar intrigante no início, mas pode vir a se tornar um fardo quando os sócios conscientizam-se de quanto esforço (presença coerente e investimento contínuo nas discussões na sala de aula ou no ambiente de trabalho) é requerido nos emparelhamentos.

C.7.1 Instrutores ou gerentes de projeto devem

- Reforçar a importância e os benefícios de *Pair Programming* com o objetivo de incentivar o comprometimento dos sócios;
- Prover tratamento especial aos membros que se mostrarem não receptivos às diretrizes de *Pair Programming* – discutir o custo benefício de *Pair Programming* e mostrar a importância da socialização promovida pela técnica pode ajudar na sua aceitação;
- Verificar, por observação, se os membros das duplas compreenderam bem as regras de *Pair Programming* (quanto à troca de papéis, quanto à natureza ativa de cada papel e quanto ao rodízio dos pares).

C.7.2 Estudantes ou desenvolvedores devem

- Compreender os benefícios advindos de *Pair Programming* (troca de informação e conhecimento, realização de software de maior qualidade, socialização);
- Ser receptivos e procurar dar uma chance ao processo de emparelhamento. Depois de algumas seções, é mais fácil concluir se os emparelhamentos estão sendo úteis ou não;
- Compreender que suas recompensas serão recebidas nos exames avaliativos individuais ou no final do projeto como consequência do seu sucesso e do sucesso da dupla.

C.8 DIRETRIZ 8 – EMPARELHAMENTOS OBRIGATÓRIOS EM LABORATÓRIOS

Emparelhamentos obrigatórios em laboratórios asseguram que uma quantia razoável de tempo será despendida em dupla na tarefa e ajudam os instrutores a identificar estudantes que não estão conformes às diretrizes de programação emparelhada.

C.8.1 Instrutores ou gerentes de projeto devem

- Promover seções de emparelhamentos obrigatórios em laboratórios (em ambientes acadêmicos) – esperar que os alunos trabalhem emparelhadamente somente fora das salas de aula prática (em atividades extra classe) pode ser temeroso. Seções obrigatórias em laboratórios podem ajudar os estudantes a se tornarem mais receptivos e mais hábeis na prática de *Pair Programming*;
- Compor pares que estão alocados a uma mesma seção de laboratório (em ambiente acadêmico) – se os sócios das duplas não estiverem lotados na mesma seção de laboratório, o emparelhamento ocorrerá somente em atividades extra classe.

C.8.2 Estudantes ou desenvolvedores devem

- Procurar resolver as atividades propostas sempre de maneira emparelhada – a não ser que a atividade seja trivial. Mesmo assim, como dito anteriormente, as atividades realizadas individualmente devem passar por processo de revisão, executado pelo outro sócio, mais adiante, antes de finalizar a atividade;
- Evitar faltar às seções de emparelhamento para não quebrar o ritmo de trabalho da dupla – a ausência de um sócio afeta diretamente as atividades do seu parceiro.

C.9 DIRETRIZ 9 – COMPRIMENTO ESPERADO DAS TAREFAS

Adaptar o comprimento esperado das tarefas, tal que uma porcentagem razoável da classe possa terminá-las no período definido para uma seção de emparelhamento em laboratório, pode diminuir a possibilidade de que sejam realizadas individualmente.

C.9.1 Instrutores ou gerentes de projeto devem

- Produzir tarefas cujo tempo necessário para solução, pelos alunos, não ultrapasse o tempo disponível a uma seção de laboratório – Isso pode, frequentemente, envolver quebras de tarefas complexas em simples (sub-tarefas);
- Incentivar os alunos a dividir as atividades extra classe propostas, seguindo os conceitos de modularização, com o objetivo de resolvê-las em encontros emparelhados pré-definidos de duas ou três horas de duração (objetivando-se evitar a exaustão). Estes encontros podem ser realizados nos finais de semana.

C.9.2 Estudantes ou desenvolvedores devem

- Explorar bem o tempo disponível durante as seções de laboratório no intuito de conseguir realizar as tarefas naquele prazo;

- Prover retorno (*feedback*) ao professor, principalmente nos casos em que o período estipulado à uma seção não foi suficiente para solução do problema proposto.

C.10 DIRETRIZ 10 – PADRÃO DE CODIFICAÇÃO

Programadores possuem estilos distintos de desenvolver suas atividades. Este diferencial pode ser prejudicial ou benéfico ao emparelhamento – Se por um lado, na falta de padrões de codificação, os sócios podem compreender estilos diferentes e aprimorar seus próprios estilos, por outro, a instituição de um padrão pode tornar o processo de emparelhamento mais ágil.

Padrão de codificação é fundamental quando se pretende instaurar *Pair Programming* na indústria de software. Porém, se o objetivo do emparelhamento de programadores envolver a troca de conhecimentos no meio acadêmico, padronizações podem ser consideradas opcionais.

C.10.1 Instrutores ou gerentes de projeto devem

- Quando em ambientes acadêmicos, instaurar seções iniciais de emparelhamento aleatório sem o uso de padrões de codificação, no intuito de incentivar a troca de conhecimento sobre os estilos de programação e, no futuro, sugerir que os alunos definam um estilo único a ser respeitado por todos;
- Quando em ambientes profissionais, conscientizar os desenvolvedores da importância em se respeitar os padrões no intuito de aumentar a qualidade dos softwares, tornando mais fácil a sua manutenção.

C.10.2 Estudantes ou desenvolvedores devem

- Procurar compreender as diferenças entre os estilos dos colegas de sala e o seu próprio estilo;
- Discutir, nas seções iniciais de emparelhamentos e em atividades extra classe, sobre o que seriam boas regras de estilo de programação.

C.11 DIRETRIZ 11 – TROCA DE CONHECIMENTO ENTRE SÓCIOS

Se um sócio conhece melhor um processo ou instrução, é perfeitamente saudável que ele use parte do tempo para ensiná-lo ao seu par.

C.11.1 Instrutores ou gerentes de projeto devem

- Incentivar a troca de conhecimento entre os sócios – enfatizar o senso de responsabilidade de cada sócio para com o seu parceiro e para com o projeto pode diminuir a resistência daqueles programadores experientes em ensinar suas técnicas e habilidades aos seus pares.
- Conscientizar os sócios de que a troca de conhecimentos é um dos maiores benefícios que a programação emparelhada agrega ao processo de desenvolvimento de software e ao processo de ensino-aprendizagem de programação de computadores;
- Conscientizar os sócios da importância em conhecer as diversas nuances de um sistema por intermédio da troca de conhecimentos proporcionada pela prática de *Pair Programming*.

C.11.2 Estudantes ou desenvolvedores devem

- Enquanto navegadores, solicitar que o piloto os ensine as instruções ou pensamentos

lógicos por eles desconhecidos;

- Enquanto pilotos, perguntar constantemente ao navegador se está compreendendo seu raciocínio. Em caso negativo, o piloto deve pausar o desenvolvimento do código e promover explicações detalhadas sobre os pontos não compreendidos pelo parceiro.

C.12 DIRETRIZ 12 – SUCESSIVOS EMPARELHAMENTOS ALEATÓRIOS

Promover sucessivas seções de emparelhamento aleatório aumenta a possibilidade de compatibilidade e permite que os sócios conheçam melhor seus colegas de turma ou de trabalho.

C.12.1 Instrutores ou gerentes de projeto devem

- Promover sucessivas seções de emparelhamento aleatoriamente até que os participantes escolham seu parceiro mais compatível;
- Conscientizar os estudantes de que diversos emparelhamentos aleatórios são ideais para criar um clima amistoso em sala de aula.

C.12.2 Estudantes ou desenvolvedores devem

- Compreender a importância em conhecer todos os colegas de turma ou todos os membros de equipes de desenvolvimento;
- Procurar compreender estilos novos de programação, mesmo que percebam dificuldades em adaptação a outros estilos.

C.13 DIRETRIZ 13 – LIBERDADE DA TROCA DE PAPÉIS

Pode ser permitido que os sócios definam o melhor momento de trocar seus papéis, porém, é importante estar consciente de que o uso desta prática aumenta o risco do piloto e do navegador permanecerem constantemente no mesmo papel, o que não seria saudável ao processo de *Pair Programming*.

É interessante também compreender que, em muitos casos, parece não haver uma divisão consistente de trabalho entre o piloto e o navegador. Ao invés disso, os dois programadores movem-se de tarefa a tarefa juntos, considerando e discutindo problemas no mesmo nível em termos de estratégia ou abstração. Isto difunde a teoria de que a passividade pode não estar diretamente relacionada ao simples ato de dominar ou não o teclado, mas às atitudes dos sócios quando se comunicam e quando há conflito.

C.13.1 Instrutores ou gerentes de projeto devem

- Permitir que os sócios definam o melhor momento de trocar seus papéis;
- Mostrar aos sócios a importância da troca de papéis para que não se tornem passivos;
- Incentivar o conflito entre os sócios, geralmente motivado pelas dúvidas e questionamentos do navegador.

C.13.2 Estudantes ou desenvolvedores devem

- Enquanto navegador, comunicar ao piloto os momentos em que deseja a troca de papel;
- Definir uma quantia de tempo a ser respeitada entre cada troca de papel;
- Compreender que a natureza do navegador pode ser ainda mais ativa do que a do piloto – a partir da observação de um (navegador), potencialmente, surgem diversos questionamentos às ações do outro (piloto);
- Evitar manter-se indefinidamente em um único papel.

C.14 DIRETRIZ 14 – MOTIVAÇÃO DOS SÓCIOS

Sócios motivados tendem a emparelhar melhor e resolver suas tarefas com maior satisfação. É importante compreender a existência de sócios desinteressados e que, ainda que colocados em trabalhos individuais, eles provavelmente não achariam motivação para trabalhar – Neste sentido, os instrutores e gerentes devem compreender que o seu papel de motivador é que determinará se os sócios irão manter-se comprometidos com as tarefas, não a natureza da atividade.

C.14.1 Instrutores ou gerentes de projeto devem

- Motivar os sócios por intermédio de atividades desafiadoras e/ou premiações;
- Procurar tornar os ambientes descontraídos – o uso discriminado de pequenas seções de piadas pode ser útil;
- Premiar duplas que apresentarem boa capacidade de comunicação e coordenação – implica em acompanhar as interações das equipes constantemente;
- Procurar membros desmotivados e estudá-los no intuito de construir políticas específicas de motivação.

C.14.2 Estudantes ou desenvolvedores devem

- Compreender, em todos aspectos, o potencial da programação emparelhada;
- Procurar motivar seus parceiros e chamá-los para a responsabilidade;
- Prover retorno aos instrutores ou gerentes quando não estiverem se sentindo motivados.

C.15 DIRETRIZ 15 – FEEDBACK DE COMPORTAMENTO

Em geral, observar o trabalho dos sócios em pares e fornecer o retorno sobre suas habilidades cooperativas não parece ser uma prática comum na pedagogia de programação emparelhada.

C.15.1 Instrutores ou gerentes de projeto devem

- Prover retorno aos sócios em relação ao comportamento e efetividade da dupla no processo de programação emparelhada;
- Acompanhar os emparelhamentos de perto e procurar verificar pontos falhos e pontos de sucesso na comunicação entre os sócios;
- Desenvolver uma lista de comportamentos falhos durante emparelhamentos e prover retorno generalizado aos seus participantes – este tipo de retorno evita que os sócios se tornem resistentes em relação ao processo, ao instrutor ou ao gerente do projeto.

C.15.2 Estudantes ou desenvolvedores devem

- Usar o retorno proporcionado pelos instrutores ou gerentes de projeto no intuito de melhorar suas habilidades de comunicação e coordenação de atividades;
- Discutir com seu parceiro sobre que comportamentos falhos listados pelos instrutores ou gerentes de projeto se aplicam a dupla e sobre como corrigi-los.

C.16 DIRETRIZ 16 – RODÍZIO DOS PARES

É importante compreender que o rodízio dos pares deve ser usado pensando-se nos objetivos para os quais ele se justifica. Por exemplo, rodízio é indicado quando for

incentivado no intuito de promover troca de conhecimento ou no intuito de compor pares compatíveis. Porém, depois de compostos pares compatíveis, e se este for o objetivo principal, o rodízio deve ser interrompido.

C.16.1 Instrutores ou gerentes de projeto devem

- Promover rodízio dos pares se o objetivo envolver a troca de informação e conhecimento sobre o sistema ou a transmissão de habilidades;
- Tornar o rodízio obrigatório quando o parceiro tiver dificuldades de programação insuperáveis ou se o mesmo não comparecer repetidamente às seções – isso diminui a possibilidade de um membro da dupla ser repetidamente prejudicado pela ausência ou omissão do seu companheiro, por outro lado, pode contribuir para que o sócio com dificuldades não consiga tirar proveito duradouro de conhecimentos sólidos por não ser hábil em compreender estilos e lógicas diferentes em um curto período de tempo;
- Levantar informações junto aos sócios sobre a efetividade por eles percebida em relação ao rodízio dos pares;
- Usar o retorno proporcionado pelos sócios para implantar políticas efetivas de rodízio.

C.16.2 Estudantes ou desenvolvedores devem

- Compreender os benefícios do rodízio dos pares (Troca de conhecimentos, socialização, procura de pares compatíveis, etc.);
- Tirar proveito do rodízio procurando em seus sócios o conhecimento de novas habilidades;
- Prover retorno aos instrutores ou gerentes de projeto sobre a efetividade dos rodízios.

C.17 DIRETRIZ 17 – PENSAMENTOS POSITIVOS

Conscientizar os sócios sobre a importância dos pensamentos positivos é um bom caminho para a motivação – por exemplo: “somos capazes”, “não há nada impossível”, entre outros. Ao pensar negativamente o tempo todo, o cérebro considera o pensamento como verdadeiro. É muito difícil trabalhar com alguém que tem insegurança ou ansiedade sobre as suas habilidades de programação. Também, pensamentos negativos como “eu sou um programador fraco” e “eu estou emparelhado com um perdedor total” destroem a relação colaborativa.

C.17.1 Instrutores ou gerentes de projeto devem

- Demonstrar com exemplos o que significa ser positivista – entrar e sair do ambiente de desenvolvimento (sala de aula ou empresa) sorrindo, pode ajudar a criar um clima harmonioso e positivista;
- Prover reforços (retorno) positivos aos sócios em relação às suas habilidades colaborativas e habilidades técnicas;
- Demonstrar confiança na capacidade dos sócios.

C.17.2 Estudantes ou desenvolvedores devem

- Esperar se beneficiar ou desfrutar da experiência de *Pair Programming*;
- Se desvencilhar de qualquer ceticismo – desenvolva uma expectativa de sucesso e cumprimente seu sócio dizendo, “Melhore-me!” Ou “Aumente meu potencial!”. Esta é uma oportunidade sem precedentes para a superação.

C.18 DIRETRIZ 18 – EXTINÇÃO DO EGOCENTRISMO

Sócios egocêntricos são resistentes à colaboração. Porém, concordar sempre com os parceiros pode ser o caminho mais curto à complacência.

C.18.1 Instrutores ou gerentes de projeto devem

- Avaliar as atividades colaborativas de acordo com requisitos colaborativos, e as demais atividades, com requisitos individuais;
- Envidar esforços para promover extinção do egocentrismo;
- Envidar esforços para promover extinção da complacência;
- Conscientizar os sócios de que atitudes egocêntricas são, na maioria dos casos, inaceitáveis em ambientes reais de desenvolvimento de software.

C.18.2 Estudantes ou desenvolvedores devem

- Procurar verificar o quanto são egocêntricos e combater suas atitudes de egocentrismo veementemente;
- Compreender que atitudes egocêntricas tornam os sócios cada vez mais defensivos em relação às idéias de outros;
- Ao trabalhar com sócios egocêntricos, mostrar que o egocentrismo do parceiro está, quando for o caso, degradando o processo de colaboração.

C.19 DIRETRIZ 19 – INCENTIVO À HUMILDADE

Incentivar a humildade pode ser um bom caminho para que os sócios compreendam

que não são infalíveis e aprendam a valorizar a opinião de seus parceiros.

C.19.1 Instrutores ou gerentes de projeto devem

- Conscientizar os sócios de que o erro é característica humana e, portanto, tomadas as devidas precauções, deve ser encarado como uma oportunidade para aprender o “como acertar”;
- Ensinar aos sócios sobre como valorizar a opinião de seus parceiros;
- Incentivar a humildade entre os sócios com o uso de dinâmicas de grupo específicas para tal.

C.19.2 Estudantes ou desenvolvedores devem

- Compreender que a humildade não deve ser encarada como fraqueza. Em ambientes colaborativos de qualquer natureza, ela é compreendida como uma das maiores virtudes;
- Compreender que ninguém é infalível e está acima das sugestões do seu parceiro.

C.20 DIRETRIZ 20 – PERÍODO DE TEMPO DEDICADO À PAIR PROGRAMMING

Especificar uma exigência de programação emparelhada de modo que nenhum programador ocupe menos de X % do seu tempo dedicado à tarefa em contato com seu sócio pode garantir que um bom percentual de tempo será gasto colaborativamente na tarefa. Quanto maior o valor de X, mais bem sucedida será a prática do emparelhamento e maiores serão os seus benefícios.

C.20.1 Instrutores ou gerentes de projeto devem

- Em ambientes de sala de aula teórica, aplicar exercícios que deverão ser resolvidos de forma emparelhada, ainda que não sejam problemas de programação – Isso ajuda a criar a cultura do emparelhamento;
- Ocupar uma porção inicial (mesmo que pequena) das aulas teóricas e práticas descrevendo estudos empíricos sobre *Pair Programming* ou estudos de caso reais observados na indústria de software – Isto requer leitura constante de artigos e periódicos específicos em *Engenharia de Software*;

C.20.2 Estudantes ou desenvolvedores devem

- Procurar material sobre *Pair Programming* disponível na Internet, principalmente os artigos apresentados em congressos e aqueles publicados em revistas especializadas – O hábito da leitura sobre o assunto torna o desenvolvedor mais familiarizado e menos resistente;
- Procurar dedicar pequenos intervalos de tempo disponível extra classe à atividade de programação emparelhada, mesmo com colegas de outras turmas (mais, ou menos avançados);
- Procurar terminar as atividades em tempo hábil emparelhado de modo que não sejam finalizadas individualmente.

C.21 DIRETRIZ 21 – EMPARELHAMENTOS EM PROJETOS

Emparelhamentos em projetos podem ajudar os participantes a entenderem o raciocínio de seus sócios, proporcionando troca de habilidades lógicas, compartilhando forças e diminuindo fraquezas.

C.21.1 Instrutores ou gerentes de projeto devem

- Conscientizar os sócios de que a atividade de projeto é um ótimo meio para a troca de habilidades lógicas;
- Conscientizar os sócios de que bons softwares são consequentes de bons projetos – o emparelhamento exclusivo da atividade de codificação só irá tornar mais claras as falhas potenciais daqueles projetos desenvolvidos individualmente;
- Incentivar os sócios a intensivamente trabalhar de maneira emparelhada também no projeto.

C.21.2 Estudantes ou desenvolvedores devem

- Requerer que seus sócios os ajudem a projetar as soluções – projetar uma solução envolve criar algoritmos, estabelecer padrões de codificação e elaborar planos de testes no intuito de se antecipar às possíveis falhas. Estas atividades, potencialmente, proporcionam a troca de conhecimentos e habilidades lógicas;
- Se possível, comparar a própria solução com as de outras duplas – pode ajudar no conhecimento de soluções mais elegantes ou mais eficazes.

C.22 DIRETRIZ 22 – DEPENDÊNCIA DO SÓCIO

A dependência dos sócios em seus parceiros torna-os cada vez menos hábeis em atividades de programação. Nestes casos, à medida que os softwares se tornam mais complexos, os sócios também se tornam cada vez mais dependentes.

C.22.1 Instrutores ou gerentes de projeto devem

- Desencorajar a dependência dos sócios e conscientizá-los da necessidade de se tornarem capazes de resolver os mesmos problemas individualmente;
- Conscientizar os sócios de que a vida profissional, em alguns momentos (talvez na maior parte do tempo), exigirá deles conhecimentos que serão aplicados individualmente;
- Procurar detectar, nas seções de emparelhamento, sócios dependentes – uma intervenção pedagógica nestes casos pode incentivá-los à mudança de comportamento.

C.22.2 Estudantes ou desenvolvedores devem

- Compreender que seus sócios os ajudam a amplificar seus talentos e habilidades, porém, não são a única fonte de conhecimento;
- Compreender que a dependência e a falta de habilidades lógicas formam um círculo vicioso. Quanto mais dependente for um sócio, menos hábil ele será e vice-verso – duas dicas para reduzir ou eliminar a dependência entre os sócios envolvem: (1) o sócio mais hábil deve solicitar explicações do sócio dependente sobre aspectos do projeto e do código produzido; (2) o sócio mais hábil deve solicitar idéias do sócio dependente sobre como resolver os problemas antes de propor suas próprias soluções.
- Procurar resolver outras tarefas individualmente, em atividades extra classe, no intuito de medir e aprimorar a própria capacidade.

C.23 DIRETRIZ 23 – INTERVALOS PERIÓDICOS

Porque os programadores emparelhados têm que se manter focalizados continuamente um no outro e na tarefa, *Pair Programming* pode ser uma experiência muito intensa e mentalmente exaustiva. Um intervalo periódico é importante para manter a força de programação emparelhada produtiva.

C.23.1 Instrutores ou gerentes de projeto devem

- Incentivar que os sócios se distraiam em períodos curtos de tempo para tomar fôlego;
- Solicitar que os participantes dos emparelhamentos pausem suas atividades momentaneamente enquanto curiosidades, notícias ou até mesmo piadas são verbalizadas – obviamente, estas intervenções devem ser cuidadosamente planejadas e não devem ocorrer a todo instante. Sob este aspecto, é interessante que os gerentes de projeto ou os instrutores acompanhem de perto o andamento da atividade para detectar o momento ideal às intervenções;
- Conscientizar os sócios de que a qualidade do trabalho deve ser priorizada em relação à velocidade de execução da tarefa e que a exaustão os encaminha ao desenvolvimento de soluções ineficazes ou ineficientes.

C.23.2 Estudantes ou desenvolvedores devem

- Interromper as atividades quando perceberem que o processo não está fluindo satisfatoriamente devido ao cansaço – os sócios podem se ocupar de outras atividades, ainda que fora do propósito de desenvolvimento do sistema, para arejar os pensamentos;
- Compreender que o cansaço mental prejudica qualquer atividade de produção.

C.24 DIRETRIZ 24 – AVALIAÇÃO DOS TRABALHOS DE OUTRAS DUPLAS

Em ambientes acadêmicos, uma outra atividade colaborativa pode ser feita depois que as atividades de programação forem finalizadas. A cada par é dado uma ou mais atividades para avaliar de acordo com os critérios dados pelo instrutor (os nomes (s) dos alunos devem ser removidos). A avaliação pode envolver a produção de um pequeno texto sobre o potencial ou carência de cada programa a ser apresentado a toda a turma. Dessa maneira, todos observarão os caminhos mais eficientes ou mais elegantes para completar uma atividade. Usualmente,

somente o instrutor tem a oportunidade de fazer essas comparações, mas elas são mais importantes para o aprendizado dos alunos.

C.24.1 Instrutores ou gerentes de projeto devem

- Incentivar avaliação de outras duplas anonimamente;
- Conscientizar os sócios de que a atividade de avaliação de outros trabalhos potencializa os ganhos de conhecimento, uma vez que lhes proporciona a interação com diversos estilos de programação e raciocínios lógicos diferenciados.

C.24.2 Estudantes ou desenvolvedores devem

- Aprender a enxergar o potencial da avaliação de outros trabalhos no sentido de influenciar o seu próprio aprendizado;
- Evitar comentários denegrindo os trabalhos de outras duplas – Um certo equilíbrio deve ser mantido entre criticar e elogiar as soluções de outrem, afinal, a verdadeira intenção de tal prática é permitir o conhecimento de outras técnicas de programação, não necessariamente melhores ou piores que as suas.