ESTUDO DE REQUISITOS PARA UM
SOFTWARE EDUCATIVO DE APOIO AO
ENSINO DA INTRODUÇÃO À COMPUTAÇÃO

Maria Clara Barros de Oliveira Fischer

Dissertação apresentada ao Instituto
de Matemática e Estatística da
Universidade de São Paulo para
obtenção do Grau de Mestre em
Ciência da Computação.

Área de Concentração: **Computação**
Orientador: **Prof. Dr. Flávio Soares Corrêa da Silva**

São Paulo, março de 2001

Estudo de Requisitos para um Software Educativo de apoio ao ensino da Introdução à Computação

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por Maria Clara Barros de Oliveira Fischer e aprovada pela Comissão Julgadora.

São Paulo, 9 de maio de 2001

Banca Examinadora:

- Prof. Dr. Flávio Soares Corrêa da Silva (Presidente) IME-USP
- Profª Drª Maria Cecília Calani Baranauskas IC-UNICAMP
- Profª Drª Heloisa Vieira Rocha IC-UNICAMP

Resumo

Este estudo apresenta a Análise de Requisitos para um software educativo de apoio ao ensino da Introdução à Computação. O caracter multidisciplinar deste trabalho fundamenta-se na Psicologia Educacional Construtivista e na Engenharia de Software. A metodologia usada baseou-se nas técnicas de Análise de Requisitos da Engenharia de Software, levando em conta as restrições funcionais de recentes paradigmas educacionais e das tecnologias computacionais.

O ensino da Introdução à Computação pode ser beneficiado pelo uso de um software educativo de apoio ao professor, propiciando aos alunos um ambiente adequado às suas necessidades individuais, onde conceitos em aprendizagem possam ser explorados e praticados. Neste estudo é considerado o histórico do uso desta ferramenta computacional, que visa aumentar a compreensão do aluno nos elementos básicos da programação de computadores levando-o a construção de uma base mais sólida de conhecimento, que irá ter como conseqüências um melhor aproveitamento nas disciplinas seguintes e uma diminuição do número de repetências.

Com base na análise levantada, este estudo apresenta uma proposta de metodologia de elaboração de exercícios baseada em três fatores: os erros mais freqüentes dos iniciantes em programação, a teoria construtivista de Piaget e os conceitos a serem aprendidos. Contudo os exercícios obtidos precisavam de um ambiente adequado à sua execução, foi então desenvolvido o protótipo de um ambiente de trabalho semi-dirigido, onde o aprendiz pode testar suas hipóteses sobre como resolver o problema, propiciando uma interação mais significativa que os atuais testes de múltipla escolha.

Abstract

This study presents a Requirements Analysis for software to support introductory courses in Computer Science. This multidisciplinary work is founded on Educational Psychology and Software Engineering. The methodology used was based on Software Engineering's Requirements Analysis, taking into account constraints of recent educational paradigms and computational technologies.

Teaching introductory Computer Science can benefit from an educational software which provides students with an environment to support individual needs. An overview of computational tools can be presented, to increase understanding of basic programming elements, leading the student to build the basic learning foundation, resulting in higher performance in future disciplines and decrease of course failures.

This study presents a methodology to elaborate exercises considering three factors: the most frequent mistakes made by a novice, concepts extracted from Piaget's Constructivist Theory and the concepts to be learned. These exercises need an appropriate environment, so a prototype of a semi-guided environment was built, where novices can test problem solving hypotheses, in a more significant interaction than with existing multiple choice test environments.

A minha querida família:

Rubens, Max e Del

Agradecimentos

Fazer uma dissertação é difícil, especialmente se você tem uma família a olhar e cuidar. No entanto, ter uma mãe como a minha, que sempre mostrou através de ações ser possível ser uma boa profissional e uma boa mãe, é um grande exemplo motivador.

Foi através do contato mais estreito com o trabalho por ela realizado que pude ter as primeiras aproximações com a psicologia educacional, despertando o desejo de estudar as possíveis uniões entre a informática e a psicologia.

Agradeço a minha grande incentivadora, minha mãe, Vera Barros de Oliveira, por toda supervisão prestada no domínio da psicologia, durante o desenvolvimento desta dissertação, com incansáveis leituras, sugestões e críticas sobre o texto.

Gostaria de agradecer também ao meu orientador, Flávio, pela sua condução desta dissertação, pelas discussões a respeito da evolução da Ciência da Computação e por me dar a oportunidade de participar do Projeto Mac-Multimídia.

E para terminar, um trecho de Lygia Fagundes Telles:

*“ ... o mesmo mar de antes, um mar que se repetia e era irrepetível,
misterioso e sem mistério ”*

Ao me deparar com esta frase, não pude deixar de pensar : o software educativo deve ser como o mar. Agradeço à escritora as belas palavras que me nortearam neste trabalho.

Índice

INTRODUÇÃO	1
1 O PROBLEMA.....	3
1.1 REVISÃO DE ESTUDOS FEITOS SOBRE ENSINO DA PROGRAMAÇÃO	4
1.2 O ENSINO FORMAL	8
1.2.1 Paradigma Antigo.....	9
1.2.2 Paradigma Novo.....	9
1.3 O ENSINO DA INTRODUÇÃO À COMPUTAÇÃO NO IME	10
1.3.1 A Posição da USP neste contexto	12
1.4 PECULIARIDADES DESSE PROJETO: PROGRAMAÇÃO ESTRUTURADA X PROGRAMAÇÃO ORIENTADA A OBJETOS.....	13
1.5 PORQUE USAR O COMPUTADOR COMO FERRAMENTA DE ENSINO?	13
2 OS RECURSOS.....	16
2.1 RECURSOS PEDAGÓGICOS: AS ESTRATÉGIAS DE APRENDIZADO.....	16
2.1.1 A Aprendizagem.....	16
2.1.2 Modelo Behaviorista.....	18
2.1.3 Modelos Cognitivos	20
2.1.4 A teoria do Desenvolvimento Intelectual de Piaget – O Construtivismo.....	21
2.1.4.1 Os Períodos do Desenvolvimento	24
2.1.4.2 Os Agrupamentos	25
2.1.4.3 O Pensamento Formal Hipotético.....	30
2.1.4.4 Operações interproposicionais com o grupo.....	31
2.1.5 Conclusões sobre os Recursos Pedagógicos	31
2.2 RECURSOS TECNOLÓGICOS	33
2.2.1 Multimídia	33
2.2.2 Hipertexto	35
2.2.3 Interface.....	37
2.2.3.1 Interação Humano-Computador.....	38
2.2.4 Internet.....	41
2.2.5 Conclusões sobre os Recursos Tecnológicos.....	42
3 O SOFTWARE EDUCATIVO COMO FERRAMENTA DE APOIO AO ENSINO.....	43
3.1 INFORMÁTICA NA EDUCAÇÃO: UM BREVE HISTÓRICO	43
3.1.1 O surgimento do Software Educacional - CAI.....	43
3.1.2 Dos Softwares generativos aos ITS.....	44
3.1.3 CAI x ITS	46
3.1.4 As Linhas de Pesquisa	47
3.1.5 Novos Rumos do Software Educacional	49
3.2 SISTEMAS Tutores INTELIGENTES (ITS - INTELLIGENT TUTORING SYSTEMS).....	51
3.2.1 Application Program Tutor – APT	51
3.2.2 Andes	53
3.2.3 Algumas Conclusões relativas aos ITS.....	55
3.3 SISTEMAS DE APOIO AO ENSINO DA PROGRAMAÇÃO PARA INICIANTES	58
3.3.1 PROUST (Program Understanding for Student)	58

3.3.2 Program Critic.....	60
3.3.3 Tutor LISP	60
3.3.4 ELM-Art	61
3.3.5 Sistema Tutor Asimov – UFPR.....	62
3.3.6 Curso de Linguagem C – UFMG	63
3.3.7 Projeto MacMultimídia – IME – USP	68
3.3.8 Algumas conclusões sobre os sistemas de apoio ao ensino da programação	71
3.4 CONCLUSÕES DESTE CAPÍTULO	71
4 A ANÁLISE DE REQUISITOS EM ENGENHARIA DE SOFTWARE	73
4.1 A ENGENHARIA DE SOFTWARE	73
4.2 ANÁLISE DE REQUISITOS E SUAS ETAPAS.....	74
4.2.1 O que são requisitos ?	76
4.2.2 Porque precisamos de requisitos ?.....	76
4.2.3 Como o analista obtém requisitos?.....	76
4.2.4 Como conduzir o processo de obtenção dos requisitos?	78
4.2.5 O Método Volere.....	79
4.2.6 O Documento de Requisitos de Software.....	79
4.3 PROBLEMAS COM OS REQUISITOS	81
4.3.1 Problemas Essenciais	82
4.3.2 Problemas Acidentais	83
4.4 MODELOS DO SISTEMA.....	84
4.4.1 Modelos Correntes.....	85
4.5 QUALIDADE EM SOFTWARE.....	87
4.5.1 Engenharia de Qualidade.....	88
4.6 SOLUÇÕES E ABORDAGENS	90
5 A ANÁLISE DE REQUISITOS PARA O PROJETO DE UM SISTEMA DE APOIO AO ENSINO DA INTRODUÇÃO À COMPUTAÇÃO	92
5.1 ANÁLISE DE REQUISITOS PELO MÉTODO VOLERE	92
5.1.1 Levantamento inicial das características do produto a ser desenvolvido	93
5.1.2 Intenções do produto	95
5.1.3 Stakeholders.....	96
5.1.4 Sistemas Adjacentes.....	98
5.1.5 Pequenos Pedacos do Software	99
5.1.6 Diagramas do Modelo Atual e Modelo Proposto	100
5.1.7 Diagrama de Contexto.....	102
5.1.8 Lista de Eventos de cada Sistema Adjacente	103
5.1.9 Lista de Eventos	104
5.1.10 Resposta do Sistema aos Eventos - Cenários.....	105
5.1.11 Observações do Serviço.....	107
5.1.12 Como usar os recursos pedagógicos na obtenção de exercícios adequados.....	110
5.1.12.1 Classes Propostas.....	111
5.1.12.2 Elaboração de Exercícios.....	114
5.1.13 Obtendo Requisitos Funcionais e Não-Funcionais.....	127
5.1.14 Protótipo dos Requisitos	132
5.1.15 Portal de Qualidade	139
5.2 DOCUMENTO DE REQUISITOS	139
5.3 CONCLUSÕES DESTE CAPÍTULO	140
6 CONCLUSÕES SOBRE ESTE TRABALHO	141

6.1 O PROCESSO DE TRABALHO E AS OPÇÕES FEITAS	141
6.2 LIMITAÇÕES	148
6.3 AVALIAÇÃO CRÍTICA – RESULTADO CIENTÍFICO	149
ANEXO I.....	150
OBTENÇÃO DE REQUISITOS - QUESTIONÁRIOS E ENTREVISTAS	150
ANEXO II	155
LEVANTAMENTO DE DÚVIDAS.....	155
DÚVIDAS SOBRE VARIÁVEIS	155
<i>Resumo:</i>	155
<i>Dúvidas Originais</i>	156
DÚVIDAS SOBRE OPERADORES E EXPRESSÕES	160
<i>Resumo:</i>	160
<i>Dúvidas Originais</i>	160
DÚVIDAS SOBRE FUNÇÕES.....	162
<i>Resumo:</i>	162
<i>Dúvidas Originais</i>	162
DÚVIDAS SOBRE COMANDOS	163
<i>Resumo:</i>	163
<i>Dúvidas Originais</i>	164
DÚVIDAS SOBRE PONTEIROS.....	165
<i>Resumo:</i>	165
<i>Dúvidas Originais</i>	165
DÚVIDAS EXTRAS (NÃO TRATADAS PELO SOFTWARE EDUCACIONAL)	165
<i>Resumo:</i>	165
<i>Dúvidas Originais</i>	165
BIBLIOGRAFIA.....	167
REFERÊNCIAS DE ENDEREÇOS ELETRÔNICOS NA INTERNET	170

Introdução

A idéia de estudar como seria um software educacional para a disciplina de Introdução à Computação, veio a partir de minha participação no Projeto Mac-Multimídia. Este projeto, desenvolvido pelo Departamento de Ciência da Computação do IME, contou com a participação de professores que lecionam esta disciplina nas diversas faculdades da USP, tendo por objetivo criar um material didático multimídia, que complementasse as aulas, facilitando a aprendizagem e motivando o aluno.

Durante o projeto ficou claro que as dificuldades eram muitas: falta de modelos de outros softwares educacionais para este domínio, diversidade de opiniões em como tratar o problema, despreparo da equipe sobre os avanços da informática na educação, entre outras.

Ao cursar as disciplinas do mestrado, descobri ser a Engenharia de Software meu ponto de maior convergência de interesses. Estava então definido o problema a ser estudado: usar a Engenharia de Software, na elaboração da análise de requisitos para construção de um software educativo de ensino da programação.

Para desenvolver esta análise baseei-me na Psicologia Educacional, onde busquei recursos adequados, estudando as teorias de aprendizado e procurando subsídios que permitissem identificar um modo eficiente de promover a aprendizagem.

Este estudo visa contribuir ao desenvolvimento da informática na educação, propondo meios para a melhoria do desempenho dos alunos em programação e diminuição do número de repetências na disciplina considerada.

O objetivo deste documento é apresentar o Estudo dos Requisitos de Software necessários para a construção de uma ferramenta de apoio ao ensino da Introdução à Computação.

Como resultado obtido ao longo do processo de análise, apresenta também uma proposta de metodologia de elaboração de exercícios baseada em três fatores: os erros mais freqüentes dos iniciantes em programação, a teoria construtivista de Piaget e os conceitos a serem aprendidos. Os exercícios obtidos precisavam de um ambiente adequado à sua execução, foi

então desenvolvido o protótipo de um ambiente de trabalho semi-dirigido, onde o aprendiz pode testar suas hipóteses sobre como resolver o problema, propiciando uma interação mais significativa que os atuais testes de múltipla escolha.

O texto está dividido em seis capítulos:

No capítulo 1 é apresentado o problema a ser estudado, justificando sua necessidade e inserindo-o num contexto social.

No capítulo 2 são estudados os recursos pedagógicos e tecnológicos necessários à resolução do problema em estudo.

No capítulo 3 é feito um breve histórico sobre softwares educacionais, dando destaque aos Sistemas Tutores Inteligentes que correspondem ao estado da arte nesta área.

O capítulo 4 contém uma introdução à Análise de Requisitos da Engenharia de Software e aos seus vários métodos.

No capítulo 5, a partir dos recursos e métodos estudados, é feita a Análise de Requisitos para a construção de um software educacional para apoio ao ensino de Introdução à Computação.

Finalmente, o capítulo 6, é dirigido a analisar criticamente as principais etapas percorridas nesta dissertação, fazendo uma revisão das escolhas feitas ao longo do processo e de suas justificativas. São discutidas também as limitações encontradas e possíveis estudos futuros.

Capítulo 1

O Problema

O problema a ser estudado é o ensino da disciplina de Introdução à Computação ministrada aos alunos do primeiro ano de diversas faculdades da Universidade de São Paulo. Esta disciplina é oferecida a várias turmas, por vários professores, resultando numa diversidade de estilos de ensino. A utilização de um software educacional neste contexto visa propiciar um ambiente de ensino-aprendizagem a ser usado fora do horário regular das aulas, exercitando os conceitos vistos em sala de aula, e possibilitando a cada aluno um aprendizado mais individualizado às suas necessidades.

Buscou-se neste trabalho focalizar o problema através de duas lentes de aproximação, a lente da Computação e a lente da Psicologia Educacional. A ótica computacional e seus métodos matemáticos nos são familiares por sua exatidão. Contudo a ótica da Psicologia ao primeiro contato nos assusta por seu caráter subjetivo, próprio das Ciências Humanas, onde não há por vezes um consenso entre seus estudiosos sobre determinado assunto, resultando em interpretações distintas do mesmo. Na Psicologia a comprovação científica de um processo não é feita através de teoremas ou provas matemáticas que certifiquem cientificamente um processo, o qual pode vir a ocorrer como a teoria prediz ou não. A força destas teorias está em sua coerência e nos experimentos e provas empíricas, onde seus resultados são comprovados dentro de uma margem probabilística de sucesso.

Inicialmente neste capítulo será feita uma caracterização do problema, através de uma revisão de estudos feitos anteriormente sob a ótica computacional, passando a seguir para a evolução ocorrida na Educação e os novos objetivos a serem alcançados, especialmente no que tange à Ciência da Computação, onde sob a ótica da teoria educacional vista, será possível nortear a fundamentação teórica para a construção de softwares educacionais.

1.1 Revisão de Estudos feitos sobre Ensino da Programação

Estudos feitos por Du Boulay e O'Shea (1981) identificaram três problemas fundamentais sobre o ensino da programação:

- I. Quais são as dificuldades encontradas pelos novatos ao aprender a programar?
- II. Quais são as características que uma linguagem de programação deve ter que a tornem mais apropriada a ser a 1^a. linguagem ensinada?
- III. Qual é a natureza da habilidade a ser ensinada e qual o melhor modo de ensinar?

Apesar de parte deste estudo estar ultrapassada, os problemas identificados continuam válidos e atuais. Veremos a seguir quais seriam as respostas atuais a cada um destes problemas.

Problema I

Quais são as dificuldades encontradas pelos novatos ao aprender a programar?

Para responder a esta questão, encontramos vários estudos feitos, inclusive contando com observações próprias feitas em minha atividade didática.

O ato de programar pode ser dividido em três etapas:

- 1) planejamento,
- 2) codificação,
- 3) testes.

Os novatos precisam desenvolver habilidades específicas a cada uma dessas etapas.

1) **O planejamento** compõe a compreensão do problema e a elaboração de uma solução. Esta solução pode ser inicialmente mental, mas já deve estar voltada a implementação, ou seja o novato deve aprender a expressar sua visão do problema como um programa. Pesquisa feita por Miller em 1974 (Du Boulay e O'Shea, pg. 149) mostrou que os alunos têm maior dificuldade com problemas envolvendo disjunções (um atributo *ou* outro) do que aqueles envolvendo conjunções (um atributo *e* outro). Na prática o planejamento escrito é pouco usado pelos alunos, seja por falta de incentivo do professor ou pelo fato dos alunos acharem

que os problemas são de fácil solução não sendo necessário descrever suas etapas (Martin, 1996). Porém a falta de planejamento acarreta um “ir e vir” mais intenso nas etapas seguintes.

2) **A codificação** envolve o conhecimento da *sintática* e da *semântica* individual de cada comando, e da *lógica de programação*- habilidade de agrupar adequadamente os comandos a fim de resultar num algoritmo (Binder e Direne, 1999). A sintaxe refere-se à escrita correta do comando, a semântica à correta aplicação deste, e a lógica ao mapeamento do problema em um programa. A habilidade semântica é normalmente considerada a mais importante e difícil de adquirir, enquanto que para a habilidade sintática é assumido que será adquirida através da familiaridade com a linguagem, das mensagens do verificador sintático e de livros texto. Na prática a maioria dos novatos possui dificuldade em corrigir erros sintáticos.

Desta forma os erros de codificação podem ser classificados como erros sintáticos, erros semânticos e erros lógicos. Erros sintáticos interrompem a compilação e produzem uma mensagem de erro, enquanto que erros semânticos e lógicos levam o programa a ‘travar’ em sua execução ou produzem dados de saída errados ou ausentes. Uma outra categoria que pode ser incluída refere-se a erros de estilo de programação que tornam um programa de difícil compreensão. Nesta categoria estão programas que usam um número alto de comandos encadeados, porém estes não serão discutidos.

A maioria dos erros cometidos por novatos são sintáticos, na proporção de 2:1 em relação a erros semânticos e lógicos, enquanto que programadores experientes cometem erros sintáticos, semânticos e lógicos na mesma proporção. Isto sugere que a 1ª. linguagem deve ter uma sintaxe consistente. Por outro lado, estudo feito por Knuth em 1971 (Du Boulay e O’Shea, pg. 152), mostra que a maioria dos programadores tende a usar um pequeno subconjunto dos construtores da linguagem, facilitando o aprendizado da sintaxe.

Alguns exemplos de erros sintáticos são:

- uso de variável não declarada;
- pontuação incorreta ou ausente (ex.: falta de parênteses, “;” ou “end”);
- erros ortográficos;
- atribuição de tipos de dados incorretos a variáveis;

- troca da posição das variáveis na atribuição;

Ex.: $a + n := \text{soma}$, quando o certo é $\text{soma} := a + n$.

Erros semânticos, no entanto são de difícil diagnóstico. Alguns dos erros frequentes são:

- uso local de variável declarada não localmente (em outra função);
- uso do comando de seleção no lugar do de interação;
- entendimento de conceitos de entrada e saída de dados;
- uso correto de expressões lógicas.

Erros lógicos mais frequentes são a integração correta de comandos e a dificuldade de acompanhar o fluxo de controle do programa

Outra prática pouco usada é a de tabular o código, facilitando a compreensão do fluxo do programa. Em estudo feito por Martin (1996), uma das vantagens da linguagem Turing sobre o Pascal é justamente o editor contar com uma função que rearranja a disposição dos comandos no editor aninhando-os adequadamente. Martin observa ainda que alguns tipos de erro foram cometidos por alunos que tinham o mesmo professor, que de certa forma levou os alunos a uma compreensão parcial do problema. Este é um bom exemplo de aplicação do software educacional, que serviria como exercitador dos conceitos expostos em sala de aula.

3) **O teste** é o processo de localizar e remover os erros da codificação. Nesta etapa programadores experientes possuem muito mais facilidade que os novatos. Os novatos têm dificuldade em compreender a lógica do programa como um todo, pois geralmente constroem o algoritmo por pedaços estanques. Na compilação, duas etapas precisam ser vencidas: erros sintáticos e erros semânticos. Os erros sintáticos são diagnosticados pelo compilador, facilitando o processo de correção. No entanto, Martin observou que mensagens mal formuladas pelo compilador levaram alguns alunos a modificar a lógica do programa, que estava correta. Já os erros semânticos e lógicos precisam de uma técnica mais apurada. Um método muito usado é a simulação do programa, seja manual ou automática; consiste em designar alguns valores de entrada para o programa e manter o registro de cada variável ao longo da execução do programa. A simulação é um método muitas vezes eficaz para a compreensão do problema, onde algumas perguntas podem ajudar:

“ Qual ação será executada para estes valores de entrada de dados ?” e

“ Sob quais circunstâncias esta ação será executada ?”

Problema II

Quais são as características que uma linguagem de programação deve ter que a tornem mais apropriada a ser a 1ª. linguagem ensinada?

Segundo Du Boulay a primeira linguagem de programação deve ser “sintaticamente simples com poucos casos especiais a serem lembrados”.

Em sua pesquisa Martin (1996) comparou as linguagens Turing e Pascal, buscando qual seria mais adequada a ser a 1ª. linguagem ensinada. Observou que uma linguagem que possua uma sintaxe mais simples, que resulte num número menor de correções é mais apropriada, pois diminui o tempo necessário a aprendizagem (neste estudo em aproximadamente 30%). Mostrou que o Turing obteve resultados empíricos melhores que o Pascal.

Segundo McCracken (In: Martin, 1996, cap2) ensinar em termos de pseudocódigo e deixar os detalhes sintáticos para os estudantes não funciona para estudantes comuns e professores comuns. As opiniões variam se os estudantes devem ser ensinados a corrigir erros sintáticos, ou devem corrigi-los sozinhos através da ajuda de livros-texto ou do ‘help’ do compilador. Muito irá depender da complexidade sintática da linguagem de programação, bem como do ambiente computacional.

O ambiente de programação deve proporcionar ajuda ao desenvolvedor, sendo tão importante quanto a própria linguagem. Um ambiente de programação deve ter o verificador de erros sintáticos integrado a um bom editor de texto. O verificador sintático fornece geralmente duas indicações: a posição do erro e o tipo de erro, embora nem sempre estejam corretas.

Problemas semânticos estão mais relacionados à perícia do novato em usar o comando, e a sua compreensão do programa. O ambiente de programação pode neste item também auxiliar ao programador, através de um debugger automático com visualização do conteúdo das variáveis, passo-a-passo do programa, possibilita a identificação de erros relacionados à

operadores lógicos nos comando de seleção ou interação, bem como a entrada e saída de dados.

No IME a linguagem adotada foi o Algol até 1990, seguido do Pascal até 1997, passando para a linguagem C, que é ensinada atualmente. Segundo um dos professores que leciona a disciplina de Introdução à Computação, o Pascal é mais didático que o C, mas o que determina a escolha da linguagem usada é a linguagem adotada pelas disciplinas seguintes. Assim se o curso contiver apenas a disciplina de Introdução a Computação, como ocorre na Matemática Pura ou na Estatística, a linguagem mais apropriada é o Pascal. O mercado de trabalho não influencia na escolha, pois não é objetivo do curso formar programadores. É ensinada apenas uma linguagem, pois segundo o professor é inviável ensinar duas linguagens ao mesmo tempo num curso introdutório.

Problema III

Qual é a natureza da habilidade a ser ensinada e qual o melhor modo de ensinar?

Para responder a este problema foi necessário conhecer um pouco das teorias psicológicas da educação. No item seguinte é mostrado como a educação evoluiu num contexto social e no Capítulo 2 veremos em detalhe as teorias psicológicas aplicadas a educação.

No Capítulo 3 é feita uma revisão dos softwares educacionais voltados para o ensino da programação, sendo destacado o estado da arte nesta área.

1.2 O Ensino Formal

A Educação vem vivendo uma época de grandes transformações. Vemos em Cortelazzo e Oliveira (1996) que os modelos usados já não atendem mais às necessidades atuais. O professor não pode se orientar ou crer no paradigma tradicional que lhe dizia *o que e como* fazer, fornecendo-lhe um conteúdo programático estável; compreende que não pode se ater predominantemente a conteúdos, constantemente revistos e ultrapassados.

“A visão restrita da Educação delimitada pela sala de aula está superada, a aprendizagem ocorre não somente num local geográfico chamado “escola” mas é um estado da mente”,

como afirma Litto (1996), o qual identifica as principais características dos paradigmas da Educação: o Antigo e o Novo

1.2.1 Paradigma Antigo

Na concepção tradicional de Educação, o aluno vem até a escola com a cabeça essencialmente vazia e cabe à escola nela colocar um conjunto de conhecimentos factuais e habilidades intelectuais, testando periodicamente a aquisição destes conhecimentos por meio de provas e exames. Neste cenário o papel ativo é exercido pelo professor; o aluno é um elemento passivo, um mero receptor dos pacotes de informação preparados pelo sistema educacional. Memorização da informação é a pedra fundamental neste paradigma. Há poucas oportunidades para aumentar a compreensão de conceitos complexos ou para estimular a imaginação. O currículo educacional é dividido em classificações estanques – matemática, inglês, geografia, sem a mais remota possibilidade de haver possíveis inter-relacionamentos entre eles.

1.2.2 Paradigma Novo

As mudanças ocorridas na sociedade brasileira nas últimas duas décadas mostram que o paradigma antigo foi superado. Entre vários fatores envolvidos, podemos destacar alguns que constituem as principais causas:

- o aumento do volume de informação disponível,
- o aumento da complexidade nos setores da vida profissional,
- a necessidade de relacionar campos do conhecimento antes isolados,
- o crescimento da necessidade de reciclagem constante,
- o aumento da internacionalização dos conhecimentos necessários para tomar decisões, para ser mais produtivo e mais competitivo no mercado de trabalho.

O novo paradigma educacional, hoje em desenvolvimento, sugere que a escola tem de ser antes de tudo, um ambiente “inteligente”, especialmente criado para a aprendizagem. Um lugar rico em recursos por ser um local privilegiado, onde os alunos podem construir seu

conhecimento segundo os estilos individuais de aprendizagem que caracterizam cada um. Na nova escola, em vez de filas de mesas e cadeiras, há mesas para trabalhar em grupo, sofás para leitura e computadores para realização de tarefas acadêmicas e trocas de comunicações. A avaliação é feita constante e serenamente na carreira do aluno. A ênfase é colocada não na memorização de fatos, mas na capacidade do aluno pensar e se expressar claramente, solucionar problemas e tomar decisões adequadamente.

O aumento do uso das novas tecnologias de comunicação leva ao uso cada vez menor do livro-texto e do quadro negro. As novas tecnologias são caracterizadas:

- pela interatividade,
- pela sua capacidade de uso individualizado,
- pela não linearidade da informação, o hipertexto é a maneira mais dinâmica e atual de apresentar a informação,
- pela capacidade de simular eventos do mundo natural e do imaginário de forma a levar o aluno a perceber fenômenos que antes não faziam parte do ensino formal por falta de apoio tecnológico que permitisse alcançar tais metas.

O papel do professor muda, agora ele passa às tecnologias de informação a responsabilidade de assegurar o aprendizado do aluno, liberando-se para ser mais um guia, um conselheiro, um parceiro na procura da informação e da verdade, aumentando a participação ativa do aluno no entendimento do conteúdo que está sendo transmitido. Sendo assim a motivação para a aprendizagem surge no aluno, de dentro para fora, ao invés de ser algo imposto pelo professor ou pelos pais. E o mais importante, há o reconhecimento de que a aprendizagem permanente daqui em diante será uma tarefa constante na vida profissional, cabendo à escola capacitar o aluno para aprender qualquer assunto que lhe interesse.

1.3 O Ensino da Introdução à Computação no IME

Vejamos agora como estes modelos se adaptam ao nosso estudo. A disciplina de Introdução à Computação, em muitos pontos, já segue o paradigma novo, beneficiando-se dos seguintes fatos:

- os computadores fazem parte integrante desta disciplina, pois são sua ferramenta de trabalho,
- a atividade de programar em si é basicamente criativa,
- a parte a ser memorizada é mínima se comparada a outras disciplinas,
- se o aluno não compreender os conceitos básicos não conseguirá escrever programas para computador.

No tangente à atuação do papel do professor, no entanto, esta permanece via de regra no paradigma antigo, onde aulas tradicionais são dadas segundo o esquema do livro-texto e do quadro-negro. O recurso tecnológico geralmente usado na aula é o retroprojeter, onde transparências sobre tópicos específicos são mostradas. As aulas restringem-se à sala, o laboratório fica disponível ao aluno, mas sem que haja a presença de um professor. Cabe ao aluno fora da sala de aula vencer as várias dificuldades sozinho, como ambientar-se ao uso da rede de computadores, ao sistema operacional, ao editor de textos e ao compilador. Este é um difícil começo, muitas vezes desmotivador ao aluno.

Vencidos os obstáculos com o ambiente computacional, o aluno pode então dedicar-se à aprendizagem da programação, porém os recursos tecnológicos não são adaptados ao aprendizado inicial da programação; o programa usado pelo aluno é o mesmo que um profissional da área utiliza, sendo que estes programas muitas vezes possuem ajuda interativa ao usuário, mas partem do princípio que os tópicos básicos lhe são familiares.

O professor em geral tem pouco contato com a produção dos alunos, pois o contato restringe-se a participação do aluno na sala de aula, dando-se este quando o aluno é chamado para auxiliar o desenvolvimento de pequenas partes do código do algoritmo que está sendo desenvolvido pelo professor. As dúvidas mais comuns neste começo muitas vezes ocorrem no laboratório, onde erros comuns de sintaxe adquirem proporções maiores que o usual, pelo fato de os alunos terem muita dificuldade de compreender as mensagens do compilador, que via de regra, usa termos técnicos em inglês. Os professores atendem muitas vezes aos alunos individualmente, fora do horário de aula, tirando dúvidas e orientando, o que supre parcialmente as carências deste modelo.

1.3.1 A Posição da USP neste contexto

A Universidade de São Paulo, ciente dos novos rumos que a Educação vem tomando, promoveu em 1998 um simpósio internacional para discutir o tema: Universidade e Novas Tecnologias – Impactos & Implicações, o qual assisti. Neste encontro foi mostrado como a tecnologia vem modificando a atividade didática nas várias faculdades da USP, seja como facilitadora do gerenciamento e organização do acervo, como no caso das Bibliotecas, seja no uso profissional, como nos exames médicos, onde os estudantes precisam conhecer todos as novidades como ressonâncias e tomografias computadorizadas, bem como usarem nas aulas práticas a videoconferência que permite a um número grande de alunos observar uma cirurgia, o que seria inviável no próprio local da cirurgia. Foram discutidos os prós e os contras do uso do computador no ensino, sendo que a principal crítica dos representantes da área de comunicação levantada foi que a interação aluno x computador, em muitos sentidos, é deficiente. Na área de computação, foram apresentados programas com interface gráfica, que permitem que experiências em Física fossem simuladas, sendo que representantes de faculdades do exterior expuseram como suas escolas vem discutindo e se adaptando às novas tecnologias de comunicação e computação. A introdução da tecnologia leva a uma mudança nos moldes de ensino correntes, levando a comunidade acadêmica a repensar o modelo. Em 1999, ocorreu um novo simpósio.

Neste novo modelo, o professor além de dominar o conteúdo a ser transmitido, deve possuir qualidades de motivação e integração com os alunos. Aulas onde os alunos reúnem-se em grupos e através da troca de experiências e dúvidas constroem seu próprio algoritmos, permitem ao professor atender individualmente aos alunos, estabelecendo uma relação mais estreita, tomando conhecimento das dúvidas que estão ocorrendo e orientando adequadamente a necessidade de cada aluno. O papel do professor deixa de ser o de transmissor do conhecimento, passando ao de facilitador, ou mediador, criando oportunidades para o aluno de ser o protagonista de seu aprendizado no processo de aprendizagem.

A construção do conhecimento pelo próprio aluno, diferente da simples memorização, é sem dúvida um processo mais lento, de descobertas, porém mais duradouro. Fornecendo menos conteúdo na sala de aula, e delegando esta função a um recurso tecnológico específico,

o professor permite ao aluno espaço para suas motivações pessoais, incentivando a individualidade da aprendizagem e favorecendo a busca e descoberta de relações interdisciplinares.

1.4 Peculiaridades desse projeto: Programação Estruturada x Programação Orientada a Objetos

Estamos vivendo uma época de transição entre paradigmas de programação. As Universidades de nosso país estão adotando gradualmente o novo modelo. Mas surge uma discussão a respeito: deve-se já adotar o paradigma da programação Orientada a Objetos - POO com os alunos do 1º ano, ou iniciar pela Programação Estruturada, e numa segunda etapa migrar para a POO?

Vejam os dois lados deste dilema. Ao se optar por iniciar pela POO, o que deve ser visto inicialmente? Os conceitos próprios deste modelo como classe, herança, etc., ou deve-se iniciar do modo tradicional pelas estruturas de programação como repetição, seleção, comparação, usando apenas o envoltório da classe?

Ao optar por iniciar o ensino pelo paradigma da Programação Estruturada, não estamos incorrendo no risco de ensinar algo que a prática vem mostrando que está superado?

Analisando mais atentamente, pode-se observar que as estruturas básicas de programação como repetição, seleção, comparação, etc., estão presentes no âmago dos dois paradigmas, constituindo uma sub-camada básica.

É sobre esta sub-camada que estaremos nos referindo neste estudo. Sendo assim, o aprendizado servirá como base aos dois paradigmas.

1.5 Porque usar o computador como ferramenta de ensino?

O grande desafio lançado à Educação, hoje em dia, por um ambiente sociocultural em profundas e rápidas transformações, é o de usar as novas tecnologias existentes como ferramenta de ensino, propiciando melhores formas de comunicação da informação e de sua compreensão.

No ensino da Ciência da Computação, obrigatoriamente iremos usar o computador como ferramenta, pois ele é o próprio instrumento de trabalho, mas como recurso tecnológico, poderíamos optar por produzir um vídeo, ou uma apresentação a respeito. Logo, é necessário justificar a adequação da ferramenta selecionada ao propósito deste estudo: atuar eficientemente no ensino da Introdução à Computação.

Inicialmente, é importante ressaltar que programas educacionais fazem do computador uma ferramenta extremamente eficaz à Aprendizagem em geral. Conforme apontado em outro trabalho de minha co-autoria (Oliveira e Fischer, 1996), o software educativo pode ser visto como um excelente instrumento na prevenção e solução de problemas de aprendizagem:

- Ele trabalha com representações virtuais de forma coerente, mas extremamente flexível, possibilitando a descoberta e criação de novas relações.
- Dispõe suas informações de forma clara, objetiva e lógica, facilitando a autonomia do usuário, favorecendo a exploração espontânea.
- Exige também que o usuário tenha consciência do que quer, se organize e informe de modo ordenado o que quer fazer, digitando corretamente.
- Dá um retorno extremamente rápido e objetivo do processo em construção, favorecendo a autocorreção, a inserção da "desordem" na ordem global.
- Trabalha com uma disposição espacial das informações, que pode ser controlada continuamente pelo usuário através de seu campo perceptivo visual, apoiando o raciocínio lógico.
- Trabalha com imagens e textos de forma combinada, ativando os dois hemisférios cerebrais.
- Através de recursos de multimídia, pode combinar imagens pictóricas ou gráficas, numa infinidade de cores e formas, com sons verbais e/ou musicais, com movimentos, criando uma trama de combinações possíveis, integrando a percepção, em suas múltiplas formas, ao raciocínio e à imaginação, de forma fluente, pessoal e cheia de vida.

Em suma, o computador por ser um instrumento programado de forma lógica e simbólica favorece o desenvolvimento de atividades mentais levando o usuário à descoberta e invenção de relações entre as informações, ou seja a organizar sistemas de forma coerente, clara e criativa.

Capítulo 2

Os Recursos

Neste capítulo veremos os recursos que dispomos para construção do software educativo. Estes recursos são de duas naturezas, os recursos pedagógicos e os recursos tecnológicos.

Os recursos pedagógicos servirão na orientação de como o conteúdo deve ser transmitido, destacando os fatores que devem ser considerados. Para compreendê-los, os dois principais modelos da Psicologia da Aprendizagem, o behaviorismo e o cognitivismo serão brevemente apresentados, e daremos um enfoque especial na Teoria do Desenvolvimento da Inteligência concebida por Jean Piaget. Deste modo, no próximo capítulo, ao acompanharmos a evolução do software educativo, poderemos entendê-lo melhor e classificá-lo de acordo com sua teoria da aprendizagem.

Os recursos tecnológicos, por sua vez, nos orientarão em como construir uma interface interativa que disponha dos modernos recursos computacionais como Internet, multimídia e hipertexto.

2.1 Recursos Pedagógicos: as Estratégias de Aprendizado

2.1.1 A Aprendizagem

O termo aprendizagem é usado freqüentemente na linguagem diária, e de um modo geral é tido como sinônimo de “conhecimento”. Neste estudo, no entanto, é necessário diferenciar o processo do produto, e sendo assim Lomônaco (1984a) ressalta que é através do processo de aprendizagem que adquirimos conhecimento.

Nos textos especializados a aprendizagem comumente é definida como uma mudança de comportamento resultante de prática ou experiência anterior. Lomônaco cita como exemplo um aprendiz de natação na primeira aula, quando o meio aquático lhe é estranho, ocorre uma descoordenação dos movimentos, aspira ar nos momentos inadequados, e executa excesso de movimentos desnecessários. O mesmo aprendiz, aulas após, quando já consegue coordenar os

movimentos e a respiração, e eliminou os movimentos irrelevantes, nos leva a dizer que este realmente aprendeu a nadar. O mesmo processo ocorre na área cognitiva quando uma criança aprende a ler e escrever. Podemos concluir que a aprendizagem resulta em uma mudança no comportamento do indivíduo.

Mas esta definição de aprendizagem restringe-se às conseqüências do processo de aprendizagem: ocorre uma mudança no comportamento, sem no entanto definir o processo em si. Este processo ocorre não externamente, mas internamente ao indivíduo. Paralelamente, o resultado deste processo (o conhecimento ou habilidade adquiridos) é externo ao indivíduo e pode ser medido, atestando o grau de aprendizagem. Desta discriminação, resulta em referir-se ao termo ‘aprendizagem’ como correspondente ao processo interno, e ao termo ‘desempenho’, como correspondente às respostas emitidas pelo indivíduo, que podem ser observadas e medidas.

Lomônaco (p. 6) finaliza com uma definição mais abrangente da aprendizagem:

“uma mudança relativamente estável num estado interno do organismo, resultante de prática ou experiência anterior, que constitui condição necessária, mas não suficiente, para que o desempenho ocorra”

Nesta definição é destacado o fato de que um indivíduo mesmo tendo aprendido algo, não necessariamente o demonstre em seu desempenho, seja por falta de motivação, ou por outro fator.

A conceituação de aprendizagem apresenta dois grupos em geral:

- a concepção de aprendizagem como mudança de comportamento defendida pelas teorias associacionistas, ou behavioristas;
- a concepção de aprendizagem como modificação do estado interno defendida pelas teorias cognitivas.

Estas correntes de pensamento possuem visões distintas não só em relação à aprendizagem, mas também quanto à aquisição da inteligência. Em linhas gerais o Behaviorismo acredita que nascemos sem inteligência, apenas com nossos instintos herdados pela espécie, e que a inteligência constrói-se ao longo da vida através de comportamentos

adaptados que são reforçados pelo meio. Por outro lado, as teorias cognitivas propõe que nascemos com a potencialidade genética de vir a pensar de forma organizada, abstrata e reflexiva, mas que esta potencialidade só se desenvolve e se converte em comportamento inteligente se o sujeito agir frente aos desafios que encontra vida à fora, de forma autônoma e criativa.

As teorias possuem muitos pontos em comum, podendo-se dizer que as teorias behavioristas e as teorias cognitivas possuem uma área de intersecção.

Estas teorias serão apresentadas no texto a seguir.

2.1.2 Modelo Behaviorista

Skinner, a partir de 1938, trabalhou em sua teoria denominada condicionamento operante, evoluindo-a então para a teoria da aprendizagem social. Pavlov em 1961, propôs uma teoria baseada no condicionamento clássico ou respondente.

O que se busca neste texto é apresentar as bases deste modelo, o que há de comum entre as diversas teorias. Sendo assim, destacamos as características principais do behaviorismo, as quais segundo Witter (1984) são:

- A quase totalidade do comportamento que caracteriza o ser humano é aprendida. Tanto o chamado comportamento ‘normal’ como o ‘anormal’ são aprendidos.
- Comportamentos ou sintomas indesejáveis constituem problemas em si mesmo, eliminar os primeiros é resolver os problemas
- Os comportamentos são mantidos por suas conseqüências ou reforços.
- Os estímulos modificam o comportamento

Os termos básicos desta teoria são *estímulo e resposta*.

Os estímulos são captados pelos sentidos, e sua presença ou ausência afeta o comportamento de um indivíduo. Estímulos ou eventos estimuladores podem ser agradáveis, neutros ou aversivos. Um determinado alimento pode ser apreciado por um indivíduo, constituindo um evento agradável a este, e ao mesmo tempo causar aversão a outro, constituindo um evento aversivo neste outro caso.

As respostas podem ser categorizadas em quatro tipos: de observação, reguladora, transacional e operante. Os tipos de resposta variam deste o virar a cabeça para melhor captar um evento, passando pela alteração do batimento cardíaco, sudorese, à operação de um equipamento, elaboração de um discurso, etc.

O behaviorista estuda como os estímulos afetam o indivíduo, observando as relações entre estímulo x resposta e suas conseqüências, também conhecido por análise de contingências.

O modelo behaviorista para aprendizagem

Em relação à aprendizagem, deve-se estabelecer quais os estímulos que promovem um aumento da probabilidade de ocorrência da resposta-alvo (estímulo reforçador), bem como ter conhecimento dos estímulos que levam a respostas inadequadas ou incompatíveis com a resposta-alvo. Os estímulos reforçadores variam basicamente de sujeito para sujeito, no entanto classes de sujeitos com características semelhantes - idade, situação mental, costumes reagem, em geral, de modo semelhante ao mesmo estímulo.

Não se pode ignorar que no início da aprendizagem ocorre um período de adaptação, onde fatores emocionais interferem no processo. Caso o aprendiz mostre um padrão de desempenho muito abaixo da resposta-alvo, são necessários novos estímulos auxiliares que reforcem o condicionamento, encaminhando o aprendiz para a obtenção da resposta-alvo. Depois que o padrão de desempenho foi atingido, é necessário mantê-lo através de um esquema de reforçamento intermitente.

O behaviorismo proporcionou uma grande produção de materiais educacionais, seja com o uso de papel ou com o uso de computadores. A partir de Skinner, foram criadas as instruções programadas, em sua versão para computador ficou conhecida por CAI (Computer Assisted Instruction) . Características deste tipo de programa, segundo Mattos (1987) são: respeito ao ritmo próprio de cada aluno e fornecimento imediato de feedback. Keller idealizou um Sistema Personalizado de Ensino, o Curso programado.

2.1.3 Modelos Cognitivos

Sob a denominação de teorias cognitivas, existem diferentes vertentes, algumas conflitantes, segundo Lomônaco (1984b). Tal fato deve-se em parte ao carácter subjetivo deste modelo, não sendo possível aferir diretamente os resultados de experimentos destas teorias, como ocorre com o behaviorismo. Todavia, Lomônaco identificou alguns pontos básicos que são reconhecidos como característicos deste modelo:

- pretende ser uma teoria abrangente ou inclusiva, explicando os fenômenos complexos da aprendizagem;
- utiliza tanto dados comportamentais quanto dados subjetivos;
- não se restringe em explicar os fenômenos baseando-se apenas no conhecimento do organismo, realça a importância das crenças e percepções da pessoa na explicação do processo de aprendizagem
- concebe a atuação do cérebro como a de combinar ou integrar a informação que lhe chega em *todos funcionais* maiores, e então operar em termos destas totalidades, ou seja a função cerebral está organizada em todos funcionais significativos, compostos por unidades menores

Uma vez definidas as características gerais desta teoria, é preciso apresentar casos individuais, particulares deste tipo de modelo de aprendizagem. Passemos então ao exame específico de uma teoria cognitiva: Aprendizagem por Insight.

Aprendizagem por Insight

Nesta teoria, um sujeito ao se defrontar com um problema ou dificuldade, vai tentar resolvê-lo de vários modos possíveis, mas a solução não é obtida diretamente ou facilmente, requer do sujeito um considerável esforço mental. Através da reestruturação perceptual da situação, ou seja quando o sujeito percebe uma nova configuração perceptual, é que a solução do problema emerge (Lomônaco, 1984c). O *insight* ocorre neste momento. O termo insight pode ser traduzido por ‘introvisão’ ou ‘visão interior’, realçando-se o aspecto de um processo interno de ver claramente as coisas.

Em Lomônaco (p.34) encontramos as explicações:

“... o insight aparece quando um homem ou um animal aprende integralmente os princípios implicados na execução de uma tarefa ou vê imediatamente as relações corretas.”

“O indivíduo que tem um insight vê a situação total de uma nova maneira, uma maneira que inclui compreensão das relações lógicas ou percepção das conexões entre meios e fins.”

O insight se caracteriza por dar uma sensação de realização, de se ter efetivamente chegado à solução do problema.

Mas como se dá o conhecimento? Quais são os processos internos envolvidos? Como o indivíduo pensa? Para tentar explicar isso buscaremos auxílio na teoria de Piaget. No entanto é fundamental ter em mente que Piaget estudou o *desenvolvimento intelectual*, procurando determinar as diversas fases que o compõe e suas características marcantes; estava essencialmente preocupado com o processo de busca do conhecimento, e não apenas com seu resultado.

A seguir será brevemente apresentada a teoria piagetiana do desenvolvimento intelectual.

2.1.4 A teoria do Desenvolvimento Intelectual de Piaget – O Construtivismo

Piaget foi um estudioso da Epistemologia Genética, isto é do processo de busca do conhecimento. As aplicações de seu estudo ao processo de ensino-aprendizagem derivaram de suas pesquisas sobre como a criança e o adolescente desenvolvem sua inteligência para se adaptarem melhor à realidade. Estudou como ocorre o desenvolvimento da inteligência no sujeito epistêmico, ou seja, não se preocupou com características pessoais individuais, mas considerou um sujeito em geral.

Em linhas gerais a teoria piagetiana diz, segundo Oliveira (1998) que a inteligência se desenvolve paulatinamente desde o nascimento começando a atingir seu patamar formal por volta dos 12 anos. A inteligência é construída através das interações do sujeito com o meio que o cerca. O sujeito busca compreender as coisas, os objetos, sendo esta uma característica intrínseca de nossa espécie. Porém o objeto oferece resistência, não é assimilado passivamente. O esforço do sujeito para entender este novo objeto é chamado de *acomodação*,

que no entanto não tem sentido de passividade, mas ao contrário, constitui um esforço interno visando aprender algo. O processo de acomodação modifica as estruturas mentais fazendo com que fiquem mais flexíveis, ágeis, abertas e abstratas.

A flexibilização das estruturas mentais favorece a descoberta das propriedades das *leis dos agrupamentos* (a partir dos 7 anos) e do *pensamento hipotético-dedutivo com os grupos algébricos* (a partir dos 12 anos) - detalhados a seguir. Neste processo o novo objeto é classificado e seriado e assim inserido na estrutura mental atual do sujeito, ou seja, o novo objeto é incorporado, ou *assimilado* a esta estrutura mental, expandindo-a. Quando a assimilação ocorre, o indivíduo é capaz de organizar melhor a realidade, compreendendo as relações e propriedades do objeto assimilado em relação aos demais objetos, já seus conhecidos. Neste processo de busca há esforço e prazer, pois a assimilação dá ao sujeito uma sensação de realização, de bem-estar.

O processo é cíclico, pois o entendimento de um objeto leva o sujeito a questionar-se sobre novas hipóteses, possíveis agora devido à luz que o recente aprendizado lançou sobre os demais objetos de sua estrutura mental e de sua realidade (Figura 1).

Flavell (1975) ressalta que a teoria de Piaget pode ser aplicada à aprendizagem, pois revela muito sobre a natureza do sujeito do conhecimento e sobre o processo pelo qual *elementos externos e desconhecidos tornam-se elementos internos e conhecidos*. Serve como um indicador das condições mais favoráveis à aprendizagem, um guia sobre a maneira pela qual se deve ensinar.

Figura 1 - O processo de aprendizado segundo Piaget

Neste prisma, segundo Flavell , devemos considerar dois aspectos:

- o problema específico de verificar como – através de que processo interno e na presença de que condições ambientais externas – o sujeito constrói as várias formas de conhecimento que vai desenvolver em crescente nível de abstração, de acordo com as pesquisas realizadas por Piaget (as conservações, o princípio de transitividade, etc.).
- o problema do ensino em sala de aula: refletindo de forma crítica, e contribuindo inclusive com recomendações concretas e específicas sobre métodos de ensino.

A partir desta visão, dois princípios fundamentais norteiam os métodos de ensino. O primeiro refere-se à relação entre o aluno e o conteúdo a ser aprendido. Deve ser propiciado ao aluno a oportunidade de participar de atividades, inclusive concretas, em seus vários níveis, relacionadas ao conteúdo. O segundo refere-se a relação entre os alunos e o processo de aprendizagem, sendo valorizadas atividades em grupo em sala de aula.

O primeiro princípio baseia-se no fato de que as cognições (aquisições de conhecimento) ocorrem através de uma troca entre o sujeito do conhecimento e o meio. O aprendiz precisa agir, pois é através da ação que aprende. O aluno deve ser conduzido a realizar ações com os materiais envolvidos, da forma, a mais variada possível que o material permitir: manusear, experimentar, olhar sob vários ângulos, pois é através da ação, de repetidas e variadas ações, que os elementos envolvidos começam a ser interiorizados transformando-se em *esquemas interiorizados*, ou *operações piagetianas* com as propriedades de composição, reversibilidade, associatividade, etc. Resumindo, deve-se levar o aluno a discriminar as ações básicas que se relacionam com um dado fenômeno, e em seguida testá-lo, realizá-lo.

Com certeza o leitor já passou por este tipo de experiência. Ao tentar realizar uma tarefa complexa pela primeira vez, verificamos que atuamos feito autômatos, ou robôs programados, seguindo as instruções que nos são fornecidas para a execução, sem no entanto captar como os elementos envolvidos se relacionam, por mais eloquente que seja o instrutor. A princípio não é possível coordenar os elementos entre si, mas aos poucos, conforme a tarefa é organizada e sistematizada através da ação, começamos a entendê-la melhor. A partir deste momento abstraímos o processo, interiorizando-o.

O segundo princípio refere-se à importância das interações com os colegas de aprendizagem, pois é através da comparação de pontos de vista distintos, de debates, que o aluno é levado a se libertar de seu egocentrismo intelectual.

O texto a seguir apresenta brevemente as etapas que constituem o desenvolvimento da inteligência, segundo a teoria de Piaget.

2.1.4.1 Os Períodos do Desenvolvimento

1) Período Sensório-motor (nascimento até 18/24 meses)

Estágio pré-lingüístico que não inclui internalização da ação no pensamento; os objetos adquirem permanência; desenvolvimento dos esquemas sensório-motores; ausência operacional de símbolos. Termina pela descoberta e combinações internas de esquemas.

2) Período Pré-Operatório (de 2 a 7 anos)

Início das manifestações da função simbólica, através da representação significativa via linguagem, imagens mentais, gestos simbólicos, jogos simbólicos, invenções imaginativas, etc. Linguagem e pensamentos egocêntricos; incapacidade de resolver problemas de conservação; internalização das ações em pensamentos; ausência de operações reversíveis.

3) Período Operatório Concreto (de 7 a 11 anos)

Aquisição de reversibilidade por inversão e relações recíprocas; inclusão lógica; início de seriação; início de agrupamento de estruturas cognitivas; entendimento da noção de conservação de substância, peso, volume, distância, etc.; início de relacionamento das operações concretas com objetos mas não com hipóteses verbais.

4) Período Operatório Formal (acima de 12 anos)

Raciocínio hipotético dedutivo. Proposições lógicas; desenvolvimento máximo das estruturas cognitivas; grupos, matrizes e lógica algébrica aparecem proposicionais: esquemas operacionais que envolvem combinações de operações.

Neste estudo interessa-nos saber o que ocorre quando uma ação é internalizada pelo sujeito. Um aluno pode fazer uma ação mecanicamente, seguindo uma ordem do professor, sem compreendê-la; nesta situação a ação não é internalizada, e não ocorre a aprendizagem. Para que o aluno entenda a ação, deve partir dele o esforço de internalizar esta ação. Neste esforço o aluno efetua uma série de operações mentais, a fim de checar as propriedades dos elementos envolvidos nessa ação.

Um software educativo, ao apresentar uma idéia nova ao aluno – *um elemento desconhecido e externo* – para torná-lo *um elemento conhecido e interno* deve portanto propiciar condições de testar de todas as maneiras este novo elemento, explorado-o quanto as suas características, a que classe pertence, quais são suas propriedades, e como se relaciona com os demais elementos presentes em sua estrutura mental pré-conhecida. Somente desta forma este novo elemento será conectado aos demais elementos da rede mental, tornando-se um novo nó desta rede. Um software educativo deve levar o aluno a refletir sobre os resultados de suas simulações. As simulações feitas no software educativo são as ações necessárias a internalização do processo, e neste ponto a ferramenta computador está apta a prover esta situação de aprendizagem.

Piaget criou um modelo bio-matemático (Oliveira, 1998), em sua teoria levantou a hipótese de que a forma de pensar no período Operatório Concreto (7 a 11 anos) é análoga a forma dos agrupamentos algébricos.

2.1.4.2 Os Agrupamentos

Serão apresentados a seguir os agrupamentos estudados por Piaget: (Flavell, 1975; Oliveira, 1996)

Leis do Agrupamento

- 1) Lei da composição (LA1): expressa a possibilidade de composição de dois elementos em um terceiro que os contenha.

Ex.: $a + a' = b$; $b + b' = c$; etc.

- 2) Lei da reversibilidade (LA2): expressa a possibilidade de que toda operação admite uma inversão.

Ex.: $x + y = z$; $z - y = x$; etc.

- 3) Lei da associatividade (LA3): expressa a possibilidade de se conseguir o mesmo resultado através de diversas operações alternativas.

Ex.: $a + b + c = b + a + c = c + a + b$; etc.

- 4) Lei da operação idêntica geral (LA4): combinando uma operação a sua inversa ela se anula

Ex.: $a - a = 0$.

- 5) Lei da tautologia (LA5): uma unidade quantitativa reunida à sua idêntica produz uma nova quantidade, ao passo que uma unidade qualitativa reunida à sua idêntica não se altera.

Ex.: $1 + 1 = 2$.

alto + alto = alto.

Os Agrupamentos

Os agrupamentos de I a IV referem-se a operações realizadas sobre classes lógicas, e os agrupamentos V a VIII abrangem as operações realizadas sobre as relações existentes entre dois ou mais elementos ou classes.

Agrupamento I (Ag I): soma primária de classes ou encaixes simples ou inclusões, refere-se a compreensão de classe e sub-classe.

Ex.: a classe dos estudantes contém a classe dos monitores

A = classe dos monitores,

A' = as demais sub-classes pertencentes a classe dos estudantes,

B = classe dos estudantes,

+ = adição lógica (união),

- = subtração lógica (intersecção com a negação)

Lei da composição	$A + A' = B$ ou $A \vee A' = B$
Lei da reversibilidade	$A - A = 0$ ou $A \wedge \sim A = \emptyset$
Lei da associação	$B - A' = A$ ou $B \wedge \sim A' = A$

Agrupamento II (Ag II): soma secundária de classes, refere-se a compreensão de sub-classes complementares a uma classe

Ex.: a classe dos estudantes contém a classe dos monitores e a classe dos estagiários

A1 = classe dos monitores,

A1' = as demais sub-classes pertencentes a classe dos estudantes, incluindo A2

A2 = classe dos estagiários,

A2' = as demais sub-classes pertencentes a classe dos estudantes, incluindo A1

B = classe dos estudantes,

$A1 + A1' = A2 + A2' = \dots = B$

Agrupamento III (Ag III): multiplicação biunívoca de classes, refere-se a possibilidade de objetos serem classificados segundo dois ou mais critérios simultâneos.

Ex.: a classe dos estudantes contém a classe dos monitores e a classe dos estagiários; os estudantes são brasileiros ou estrangeiros.

	Monitor	Estagiário
Brasileiro	classe dos monitores brasileiros	classe dos estagiários brasileiros
Estrangeiro	classe dos monitores estrangeiros	classe dos estagiários estrangeiros

Agrupamento IV (Ag IV): multiplicação counívoca de classes, refere-se a compreensão de uma estrutura com ramificações que se multiplicam, semelhante a estrutura de uma árvore.

Ex.: a classe dos estudantes contém a classe dos monitores e a classe dos estagiários; e está contida na classe das pessoas.

Agrupamento V (Ag V): soma de relações assimétricas, refere-se a compreensão de que em uma série há sempre a possibilidade de se estabelecer uma relação inversa.

Ex.: $A < B$ e $B < C$ então $A < C$ (transitiva)

$$A < B \neq A > B$$

$$A < B \rightarrow B > A$$

Agrupamento VI (Ag VI): soma de relações simétricas, inclui composições aditivas de vários tipos diversos de relações simétricas.

Ex.: relações de parentesco simétricas, se B e C são irmãos, eles têm o mesmo pai A; se D é primo de B, então também é primo de C; de E não é tio de B, então E também não é tio de C.

Agrupamento VII (Ag VII): da multiplicação biunívicas de relações, expressa a possibilidade de trabalhar ao mesmo tempo com duas séries, buscando a correspondência segundo uma ou duas relações

Ex.: Os objetos são ordenados segundo dois atributos, peso e volume. Neste caso a representação em forma de uma matriz torna mais clara a compreensão:

- a ordenação da série em uma linha é feita pelo aumento de peso, em objetos de mesmo volume

- a ordenação da série em uma coluna é feita pelo aumento de volume, em objetos de mesmo peso

Agrupamento VIII (Ag VIII): da multiplicação co-unívoca de classes, refere-se à possibilidade de estabelecer ao mesmo tempo várias relações em várias dimensões, simétricas ou não.

Ex.:

Neste árvore genealógica:

- verticalmente: P é pai de A1 e A2,
- horizontalmente: A1 é irmão de A2,
- diagonalmente: A1 é tio de B3 e B4.

Classe e Série

Através da compreensão dos agrupamentos, o sujeito está apto a classificar e seriar. Em Oliveira (p. 32):

“...a noção de número (é construída pelo sujeito) quando, após se dar conta de que qualquer elemento do conjunto é equivalente aos demais ($1=1=1$ etc.), toma consciência da: série: $1 < (1+1) < (1+1+1) \dots$, e da inclusão de classe: $\{[(1) \Rightarrow 1] \Rightarrow \}$ etc., ou seja, ele compreende ao mesmo tempo que um é menor que dois, que é menor que três e que um está contido em dois, que está contido em três. A construção das séries e das classes supõe assim a separação e integração indivíduo-universo, elemento-todo.”

A classificação é uma busca de semelhanças entre elementos, e uma vez classificados estes elementos são seriados, através de diferenças. Neste processo o sujeito procura descobrir o que muda e o que permanece, quais as características invariantes. A *conservação* e a *transformação* são noções complementares e necessárias as operações.

2.1.4.3 O Pensamento Formal Hipotético

A entrada no pensamento operacional formal se inicia por volta dos 12 anos, e sua principal característica refere-se à distinção entre o real e o possível. A partir desta idade o sujeito já é capaz de ao se deparar com um problema, tentar imaginar todas as possíveis relações entre os elementos envolvidos, através tanto de experiências (ação sobre os elementos) como de análise lógica (abstração dos elementos). Deste processo tenta verificar quais destas relações possíveis são realmente verdadeiras, ou seja nesta idade a realidade é compreendida como um subconjunto do que os dados permitem admitir como hipótese.

O mesmo problema é visto de modos diferentes de acordo com a idade ou período de desenvolvimento que o sujeito se encontra, no período de 7 a 11 anos os indivíduos descobrem as associações de classes básicas $A \times B$, $A \times B'$, etc. a medida que experimenta, manuseia os dados; enquanto que após os 12 anos o sujeito tende a conceber estas relações *antes de experimentá-las*, e possui uma técnica para produzir todas as combinações possíveis destas associações.

Para descobrir o possível, o sujeito explora todas as possibilidades, submetendo as variáveis do problema a uma análise combinatória. Desta resulta uma rede de possibilidades hipotéticas. O pensamento formal hipotético é semelhante à lógica proposicional, onde cada dado envolvido no problema é uma *proposição lógica* (uma afirmação). As operações lógicas básicas são a negação, a conjunção e a disjunção. As classes básicas $A \times B$, $A \times B'$, $A' \times B$ e $A' \times B'$ convertem-se agora em proposições, indicadas por $p \wedge q$, $p \wedge \sim q$, $\sim p \wedge q$, e $\sim p \wedge \sim q$, respectivamente.

2.1.4.4 Operações interproposicionais com o grupo

No período formal o sujeito aplica quatro transformações (funções) às sentenças lógicas, compostas por proposições e operadores lógicos.

- 1) Identidade (I) (OpInter I): é a transformação nula, não altera em nada a sentença sobre a qual incide.

$$\text{Ex. I } (p \wedge q) = p \wedge q$$

- 2) Negação (N) (OpInter N): esta transformação modifica tudo, ou seja as afirmações tornam-se negações, as conjunções disjunções e vice-versa.

$$\text{Ex. N } (p \wedge q) = \sim p \vee \sim q$$

- 3) Reciprocidade (R) (OpInter R) : esta transformação permuta as afirmações e negações, sem alterar as conjunções e disjunções.

$$\text{Ex. R } (p \wedge q) = \sim p \wedge \sim q$$

- 4) Correlatividade (C) (OpInter C): esta transformação permuta as conjunções e disjunções, sem alterar as afirmações e negações.

$$\text{Ex. C } (p \wedge q) = p \vee q$$

2.1.5 Conclusões sobre os Recursos Pedagógicos

Foram apresentadas nesta parte do capítulo duas das principais correntes da psicologia da aprendizagem: o behaviorismo e o construtivismo. A apresentação do construtivismo foi mais detalhada pois selecionamos este modelo de aprendizagem para a construção do software educacional.

Há alguns pontos básicos vistos ao longo deste capítulo, que podemos ressaltar segundo Oliveira (1996):

- As estruturas mentais se desenvolvem à medida que interagem com o meio visando à adaptação.
- É o indivíduo o sujeito da aprendizagem buscando lidar melhor com a realidade.

- Há nesse processo uma conexão dinâmica entre mecanismos retro e pró-ativos, sendo que é no momento atual que o sujeito processa a síntese dessa articulação.
- Há portanto, uma lógica que organiza o vivido, estruturando sistemas de lembranças pessoais ao presente e ao futuro.
- No processo de auto-regulação há, ao mesmo tempo, ordem e espontaneidade, dada a capacidade de o sujeito inserir perturbações momentâneas no processo global de estruturação, à sua moda.

Este trabalho, baseado em Piaget, acredita no hard-core piagetiano que diz que as estruturas mentais são dotadas da capacidade de auto-regulação, ou seja que se desenvolvem à medida que organizam e se utilizam de informações já registradas pelo sujeito para resolver seus problemas.

Essa hipótese diretriz contém pressupostos teóricos:

Toda aprendizagem é ativa. É o sujeito que parte em busca do conhecimento daquilo que acha que falta. O conhecimento não se reduz a uma transmissão de informação de fora para dentro dada pelo professor, mas supõe uma ação mental do sujeito que aprende.

Para assimilar o conhecimento, o sujeito deve fazer um esforço interno para relacionar o novo objeto do conhecimento ao conhecimento já adquirido. Em suma, para que haja ensino/aprendizado é necessário que a nova informação seja organizada pelo sujeito (nível formal do aluno) e que ela seja significativa para os registros internos anteriores do aluno (nível de conteúdo/ tema).

A verbalização do processo em seus dois sentidos (idas e vindas), assim como do que já aconteceu e do que vai acontecer, com o registro mental dos êxitos e fracassos obtidos, leva do mero fazer por ensaio e erro ao compreender as seqüências e a utilizá-las de forma intencional, refletida e ágil. As tentativas impulsivas, os toques irrefletidos vão pouco a pouco sendo substituídos por uma execução planejada, organizada, com grande aproveitamento de indícios, utilização da memória, exclusão de movimentos desnecessários e crescente coordenação de ações.

2.2 Recursos Tecnológicos

Na última década houve uma tremenda evolução em termos de qualidade dos microcomputadores, e em especial dos monitores de vídeo, o que propiciou um avanço na qualidade gráfica dos softwares. Em paralelo o barateamento do computador incluiu-o na lista dos equipamentos indispensáveis de uma casa. A Internet, antes restrita aos meios acadêmicos e militares, com a popularização da Wide World Web- WWW permite aos usuários domésticos de computador conectarem-se maciçamente. Neste panorama, os alunos que usarão o software educativo proposto, não irão se contentar com padrões abaixo daqueles que estão acostumados, portanto a Especificação dos Requisitos irá se basear nos recursos tecnológicos mais atuais.

2.2.1 Multimídia

O computador evoluiu muito desde sua invenção. Como nos mostra Hodges e Sasnett (1993), no início sua utilização era restrita a processamentos numéricos, e alguns achavam que poucos computadores seriam necessários devido à pequena demanda existente para grandes volumes de cálculo. Mas o computador começou a ser usado para outros fins, sendo que atualmente sua utilização básica ocorre nas áreas de edição de textos, manipulação de planilhas, editoração gráfica e Internet. O computador em sua evolução tornou-se um instrumento para comunicação e expressão de idéias, uma nova tecnologia de comunicação. Seu uso deixou de ser restrito às pessoas ligadas às áreas técnicas em computação, abrangendo atualmente a sociedade em geral.

A computação multimídia corresponde a mais um passo na evolução do uso do computador como mídia de comunicação. Não há um consenso sobre o que é ou não “multimídia”. O termo é usado geralmente para referir-se ao uso de dados de imagens e sons. Mas “multimídia” é mais do que o uso de novos tipos de dados, é uma integração de mídias resultando num *documento composto*. Este processo levou a um repensar nos usos e possibilidades de integração do computador, resultando em centenas de acessórios multimídia, e novas gerações de *estação multimídia*.

Para entender melhor, é preciso esclarecer o que vem a ser ‘*mídia*’. Segundo Baitello Jr. (1998), a mídia constitui o meio de transmissão da informação. Existem vários níveis de mídia, sendo detalhados a seguir:

- *Mídia primária*: nosso corpo, pois se comunica e cria vínculos com outros indivíduos através da fala, dos gestos e ações.
- *Mídia secundária*: a escrita, amplia o círculo de recepção, nas dimensões do tempo e do espaço. A imprensa usa a mídia secundária através do livro e do jornal.
- *Mídia terciária*: duplo aparato: emissão e recepção. Nesta categoria estão o rádio, o computador, e a TV. A grande mídia da Internet é o telefone e por decorrência a eletricidade.

A mídia terciária engloba a secundária que por sua vez engloba a primária, sendo que toda mídia emergente se inicia pela mídia anterior. Assim temos como exemplo a evolução ocorrida com o jornal, a versão escrita serviu de base para o jornal transmitido pelo rádio, que por sua vez foi a base do telejornalismo. Para a Internet a base foi o jornal escrito, mas ainda não se sabe a forma final deste tipo de jornalismo. O sentido da visão é o que necessita de maior quantidade de bits para ser comunicado, assim é o nosso sentido mais desenvolvido.

Na área da Educação, os primeiros programas feitos procuraram seguir o esquema dos livros-texto, evoluindo para a interatividade atual. Sendo assim a multimídia engloba a escrita, as imagens e os sons. Outro termo comumente usado é “multimídia interativa”, onde ocorre a comunicação bidirecional entre os pontos que se comunicam: computador e usuário, criando uma interação.

Evolução da Comunicação

Nos primórdios da humanidade a forma usual de comunicação era a oral. A informação ficava armazenada na memória das pessoas. Segundo Hodges e Sasnett, com o surgimento da escrita, gradativa e lentamente, a humanidade tomou-a com forma corrente de registro da informação. Através da tecnologia da computação multimídia, após centenas de anos, estamos aptos a adotar novamente a comunicação oral como forma de transmissão da informação

registrada. Esta nova tecnologia de comunicação incorpora o uso de redes de informação interligando grandes bases de dados de textos, imagens e sons: os hipertextos.

2.2.2 Hipertexto

O hipertexto vem sendo largamente usado em domínios como aprendizado e interface com o usuário (Pagano, 1992), sendo comumente referido como uma forma não linear de visualizar a informação. O hipertexto permite que a informação desejada seja visualizada através da seleção de um tópico específico, levando a pessoa a percorrer a base de informações aos “saltos”, ao invés de ser feita a busca através de uma palavra-chave. Um documento em hipertexto pode ser impresso como um livro, no entanto, é necessário um computador para que funcione.

Um hipertexto pode ser contextualizado como uma rede de nós e ligações. Um nó ou um conjunto de nós formam um documento, e as ligações são as referências cruzadas do documento, tais como notas de rodapé, ilustrações, e até mesmo outros documentos. O conceito mais importante em hipertexto é o link (ligação). Um link conecta um nó a outro. Quando um nó é ativado, um salto é dado para o documento apontado pelo link. Pode existir um único tipo de link ou muitos em um hipertexto. Links podem ser unidirecionais ou bidirecionais, devem ser fáceis de ativar e devem ter tempo de resposta curto.

Um nó de um hipertexto é um simples documento, uma unidade de informação indivisível; o tamanho do nó, ou sua *granularidade* é uma controvérsia. Excesso de fragmentação pode tornar a exposição confusa, enquanto que uma modularização insuficiente pode tornar aborrecida a visualização de um extenso texto. Não há um consenso geral que nos guie na escolha da granularidade, mas em geral um nó deve exibir um único conceito. Uma solução é criar uma cadeia de nós, unidos em seqüência, mas permanecendo como nós.

A estrutura em rede de um hipertexto é fator determinante na facilidade de criação, uso e atualização do texto. O comportamento de um sistema em hipertexto pode ser visto nos seguintes tópicos:

- Os nós de um documento podem ser criados, eliminados, alterados e copiados.

- As janelas podem ser reposicionadas, fechadas, ter seu tamanho alterado, ou ser minimizadas.
- Os usuários navegam em hipertexto através da seleção de links que ‘viajam’ de nó a nó. O acesso a um nó é feito numa região da tela onde há uma ação associada, conhecida como “hot spot” ou “live area”, apontando o mouse nesta área e clicando, ocorre uma mudança de nó. Este mecanismo é conhecido como navegação em um “browser”.
- Os nós foram feitos com a intenção de serem vistos através de um browser, que é um mecanismo que permite navegar pela estrutura. É o responsável pelo armazenamento eficiente dos nós percorridos, tornando fácil sua obtenção, quando se deseja percorrer o caminho reverso. O documento pode ter uma estrutura complexa, especialmente se um nó for outro documento. O browser permite ao usuário acessar diretamente qualquer nó visível no ‘mapa’ do documento, mostrando ao usuário sua localização neste mapa.

Uma visão filosófica do Hipertexto

O filósofo contemporâneo Pierre Levy (1993) dedica um livro ao assunto hipertexto, mostrando-nos aspectos vistos por outro ângulo. Deste estudo, Levy caracteriza o hipertexto através de seis princípios abstratos:

- 1) *Princípio da metamorfose: a rede está em constante construção e renegociação entre os atores envolvidos, seres humanos, palavras, imagens, etc.*
- 2) *Princípio da heterogeneidade: Os nós da rede hipertextual são heterogêneos, compostos por imagens, sons, palavras; as conexões são lógicas, afetivas, etc. Na comunicação as mensagens serão multimídias, multimodais, analógicas, digitais.*
- 3) *Princípio de multiplicidade e de encaixe das escalas: a organização do hipertexto é ‘fractal’, ou seja qualquer um dos nós pode ser composto por uma outra rede, e assim por diante, indefinidamente.*
- 4) *Princípio da exterioridade: a rede não possui unidade orgânica, ou motor interno, sua composição depende de um exterior indeterminado: adição de novos elementos, ligação com outras redes, etc.*

- 5) *Princípio de topologia: nos hipertextos tudo funciona por proximidade, por vizinhança, sendo o curso dos acontecimentos uma questão de topologia, de caminhos. No hipertexto o espaço restringe-se à rede, que é o único caminho das mensagens, que não podem circular livremente.*
- 6) *Princípio de mobilidade dos centros: a rede não tem um centro, possui diversos centros perpetuamente móveis, saltando de um nó a outro, trazendo ao redor de si uma ramificação de pequenas raízes.*

O uso do hipertexto associado à multimídia torna-se uma via poderosa de interatividade entre o software educacional e o aprendiz, tornando o ambiente computacional motivador à aprendizagem.

2.2.3 Interface

A comunicação existente entre o software e o aluno, é fator relevante ao sucesso do mesmo, sendo para Galvis (1997), o terceiro eixo que compõe a natureza interdisciplinar dos softwares educacionais; os outros eixos são o educacional e o informático. Para a construção de um software educativo de qualidade, deve-se buscar alcançar uma interface amigável. A ênfase deve ser mudada de ‘*o que mostrar*’ para ‘*como apresentar*’, tendo como alvo o usuário e sua satisfação em usar o software.

Segundo Heckel (1993), os fatores a serem considerados são:

- *Analistas pensam logicamente, não visualmente:* os analistas buscam soluções lógicas, mas um bom comunicador preocupa-se em comunicar-se visualmente. O usuário espera que haja clareza nas comunicações vindas do software.
- *Analistas baseiam-se em si próprios quando pensam no usuário:* um usuário não quer entender termos informáticos, ele espera ler uma comunicação com termos que lhe sejam familiares.
- *O software avalia as ações dos usuários:* o analista em geral considera a informação fornecida pelo usuário como certa ou errada, quando o usuário não espera que suas ações

sejam julgadas pelo programa, mas que o programa o ajude a superar os problemas decorrentes de sua utilização.

- *Analistas fazem o software ter o controle da situação:* o analista ao controlar o programa, acaba tendo o controle sobre as ações do usuário, enquanto este espera ter a sensação de que controla o computador e a tarefa.
- *Analistas pensam no geral e não no específico:* o treinamento recebido pelos analistas os leva a sempre abstrair e generalizar a resolução de problemas, enquanto que a maioria das pessoas se sai melhor pensando em termos concretos e específicos.
- *Analistas pensam somente na estrutura interna do programa:* os analistas tem suas preocupações voltadas para a organização interna do programa, enquanto que o comunicador se preocupa principalmente com a estrutura externa percebida pelo usuário, e o usuário não quer que a estrutura interna do programa interfira na interface.
- *Analistas procuram simplificar a estrutura interna do programa:* enquanto que o comunicador procura fazer tarefas complexas que resultem numa interface simples para o usuário, e o usuário espera que o programa seja no máximo tão difícil de entender quanto a tarefa a ser executada, não mais do que isto.
- *O conhecimento dos analistas limita sua visão:* os analistas em relação à programação sabem o que pode e o que não pode ser feito, mas ficam receosos de implementar algo do que não conhecem todos os detalhes. O sucesso e aceitação do programa pelo usuário depende basicamente de como é a interface, sendo assim, deve-se pensar antes no resultado esperado para o programa, e depois na forma como este deve funcionar. O usuário deve preocupar-se com a tarefa que está tentando executar e não com o mecanismo necessário para a sua execução.

2.2.3.1 Interação Humano-Computador

Mais recentemente, na década de 90, alguns cursos de Ciência da Computação incorporaram à sua grade de disciplinas, matérias que pretendem entender melhor a interface com o computador, assim surgiu a Interação Humano-Computador (IHC), que é uma

disciplina preocupada com o design, avaliação e implementação de sistemas computacionais interativos para uso humano e com o estudo dos principais fenômenos envolvidos. (Rocha e Baranauskas, 2000).

A performance humana no uso de computadores depende da qualidade da interface adotada no sistema computacional. Uma interface de usuário bem elaborada proporciona, a usuários de diversos campos de atuação, uma mudança para melhor na forma de atuar. Médicos podem fazer diagnósticos mais precisos, pilotos têm mais segurança em seus vôos e alunos desenvolvem suas idéias em ambientes de aprendizagem. O uso de computadores pode efetivamente enriquecer o trabalho e a vida das pessoas.

Uma interface e sua interação não podem ser desvinculadas, são dependentes. O comportamento do lado computador é bem conhecido; mas como funciona o lado humano desta interface? Para responder a esta questão, estabeleceu-se um modelo do 'processador de informações humano'. Para construção deste modelo, foram feitos experimentos que quantificaram os tempos de resposta humano a determinados estímulos. O modelo é formado por três sistemas: perceptual, motor e cognitivo. Sendo assim, foi verificado que o tempo de ciclo do processador perceptual é de 100ms, ou seja uma imagem só é 'vista' se permanecer por um tempo mínimo de 100ms. A memória, por sua vez, está dividida em memória de curta duração e memória de longa duração, cada uma possuindo características próprias.

O objetivo da IHC é fornecer critérios que permitam a construção de sistemas agradáveis de usar, e que tenham alto grau de aceitabilidade por parte dos usuários Surgem aqui dois importantes conceitos:

- *A utilidade* verifica se a funcionalidade do sistema faz o que deveria ser feito;
- *A usabilidade* de um sistema diz respeito a correta e simples utilização do sistema em toda sua utilidade.

A usabilidade pode ser definida a partir de cinco atributos:

- 1) *Facilidade de aprendizagem*, o usuário deve ser capaz de rapidamente poder atuar no sistema computacional;

- 2) *Eficiência*, uma vez aprendida uma tarefa, esta deve propiciar um grau elevado de eficiência na produtividade;
- 3) *Facilidade de lembrar*, caso o usuário permaneça afastado do sistema por um tempo, ao tornar a usá-lo deve poder facilmente recordar sua utilização;
- 4) *Erro*, é entendido como uma ação que não leva ao resultado esperado. O sistema deve ter uma baixa taxa de erros.
- 5) *Satisfação subjetiva*, os usuários devem gostar de usar o sistema, especialmente fora do ambiente de trabalho, onde o sistema não pode ser imposto.

Para atender às necessidades de usabilidade e de uma interação adequada humano-computador, foram identificadas algumas heurísticas que norteiam o design e a avaliação de sistemas computacionais (Rocha e Baranauskas, 2000, Tabela 4.2, pag. 170):

- 1) *Visibilidade do status do sistema*, feedback do que está ocorrendo no momento com o sistema, assim se um documento é enviado para impressão queremos saber se a impressão está ocorrendo e quando termina;
- 2) *Compatibilidade do sistema com o mundo real*, a linguagem usada deve ser compreensível pelo usuário, usando termos e conceitos familiares ao usuário. A informação deve aparecer em ordem natural e lógica;
- 3) *Controle do usuário e liberdade*, usuários com frequência escolhem funções por engano, necessitam de saídas claras e sem extenso diálogo, e de funções UNDO e REDO que retornem a situação anterior ao erro;
- 4) *Consistência e padrões*, evitar de usar dois termos para a mesma situação ou ação, e seguir as convenções de plataforma computacional;
- 5) *Prevenção de erros*, é melhor evitar o erro do que querer consertá-lo através de mensagens;
- 6) *Reconhecimento ao invés de lembrança*, ações e opções devem estar visíveis. O usuário não deve ter de lembrar-se de algo que não está visível;

- 7) *Flexibilidade e eficiência de uso*, usuários novatos se tornam peritos com o tempo, o sistema computacional deve prever a evolução do usuário, e disponibilizar atalhos ou facilitadores de tarefas repetitivas, que agilizem ações frequentes;
- 8) *Estética e design minimalista*, os diálogos não devem conter informações irrelevantes, só deve ser posto o que realmente é relevante, excesso de informação diminui a visibilidade ao alvo;
- 9) *Ajudar os usuários a reconhecer, diagnosticar e corrigir erros*, mensagens de erro devem ser expressas em linguagem clara e compreensível ao usuário, indicando o problema e sugerindo uma solução construtiva;
- 10) *Help e documentação*, embora seja melhor um sistema que possa ser usado sem documentação, o help deve existir. As informações devem ser focalizadas na tarefa e breves.

A consideração destas heurísticas permite ao desenvolvedor de sistemas computacionais criar uma interação homem-computador com alto grau de usabilidade. E mais do que isto despertam a consciência no desenvolvedor de que o sistema computacional é feito não apenas para dar uma resposta correta, mas para ser usado por outro ser humano com limitações e preferências de uso.

2.2.4 Internet

Na década de 70, o Departamento de Defesa dos Estados Unidos criou a rede militar ARPANET, a Internet surgiu do aprimoramento desta rede, como nos ensinam Setzer e Kon (1995). Foi definido então o Protocolo Internet - IP que permitiu que redes locais se interconectassem. O protocolo TCP foi agregado, criando assim a rede Internet formada por computadores interligados segundo o protocolo de comunicação TCP/IP. À Internet estão conectados desde supercomputadores a notebooks, sendo o meio de comunicação composto por linhas telefônicas, cabos de metal, fibra óptica ou satélites.

Os principais serviços disponíveis aos seus usuários são:

- correio eletrônico,

- chat, troca interativa de mensagens em tempo real,
- World Wide Web - WWW, navegação em sites de hipertextos multimídia,
- discussões em grupos de interesse da USENET ou através das listas de mail
- conexão em máquinas remotas,
- transferência de arquivos.

A Internet é um meio de permitir aos alunos que fora do ambiente da escola, ou do horário de aula, possam se comunicar com o professor ou os colegas, a fim de esclarecer dúvidas. Permite também que sejam feitas pesquisas permitindo ao aluno obter informações de diversas fontes.

2.2.5 Conclusões sobre os Recursos Tecnológicos

Vimos que os principais recursos tecnológicos a serem usados serão uma combinação da multimídia com o hipertexto, buscando usar os princípios de usabilidade da IHC a fim de criar uma interface interativa e amigável onde a prioridade está na comunicação da idéia e não na melhor forma de construção do software. A Internet será usada para que haja comunicação entre os alunos e professores.

Capítulo 3

O Software Educativo como Ferramenta de apoio ao Ensino

Neste capítulo será mostrada a evolução por que passam os softwares educacionais, relacionando estas transformações segundo dois pontos de vista distintos: o da Psicologia Educacional e o da Computação, que serviram como base aos novos modelos. Uma nova concepção sobre sistemas computacionais está em formação, privilegiando o processo de comunicação humano-máquina, esta interação leva em conta aspectos ergonômicos do software. Veremos que o desenvolvimento de software educativo é um problema bem complexo, pois pode ser visto como uma modelagem do processo de transmissão do conhecimento.

3.1 Informática na Educação: um breve histórico

3.1.1 O surgimento do Software Educacional - CAI

O uso de computadores na Educação iniciou-se por volta de 1950. As primeiras tentativas de desenvolvimento de softwares educacionais apoiaram-se no behaviorismo. A partir de Skinner, foram criadas as instruções programadas. Sua versão para computador ficou conhecida por CAI (Computer Assisted Instruction), a qual proporcionou uma grande produção de material educacional. Segundo Valente (1988), a instrução programada consiste em dividir o material a ser ensinado em módulos encadeados, sendo que cada módulo contém um conceito ou fato. Ao final de cada módulo o aluno deve responder algumas questões de múltipla escolha ou preencher espaços em branco. Caso o aluno acerte a resposta passa para o próximo módulo. Caso contrário o programa mostra a resposta certa ou o aluno é levado a refazer partes do módulo atual. Foram feitos programas que auxiliam o professor na organização da seqüência das aulas, na correção de testes, mas principalmente programas que interagem com os alunos. Neste último caso o programa educacional poderia atuar

basicamente de duas formas: o aluno poderia ter mais liberdade no uso do programa, ou o programa ter o controle explícito sobre o aprendizado.

Durante a década de 60 o governo americano investiu muito na produção de CAI, seja através de empresas como IBM, Digital e RCA ou em Universidades como Standford na Califórnia. Os programas foram criados inicialmente para os cursos de matemática e gramática do 1º. grau, porém nesta época o custo do computador era proibitivo para escolas, e os alunos usavam os terminais dos computadores da universidade, o que dificultava o acesso. Em seguida foram criados programas para os cursos ministrados na própria universidade. O CAI mais conhecido e bem sucedido é o PLATO, desenvolvido no início dos anos 70 pela Control Data e a Universidade de Illinois, chegou a ser usado em 950 terminais, localizados em 140 locais diferentes.

Os computadores eram vistos mais como livros, isto é, como depositários estáticos do conhecimento contendo também perguntas e respostas pré-definidas. Contudo era o primeiro passo na individualização do ensino. O software continha limitações de acesso dinâmico ao conhecimento nele contido, não podendo portanto responder a questões não previstas pelo autor, ou adaptando-se ao estilo de busca de informação desejada pelo aluno. O programa educacional tinha, em geral, o controle explícito sobre o aprendizado, permitindo ao aluno pouca liberdade no uso do programa (Pagano, 1992).

Nos anos 70 surgiram os primeiros programas generativos, capazes de gerar automaticamente parte do material instrucional; temos como exemplo programas de ensino de aritmética onde os números eram gerados aleatoriamente e as operações resolvidas automaticamente, comparando a solução com a resposta do aluno. O ponto fraco estava no fato de o programa não poder ajudar ao aluno em compreender 'como' obter a solução correta, sabia informá-lo apenas se sua resposta estava correta ou não.

3.1.2 Dos Softwares generativos aos ITS

Carbonell e Sleeman (In: Viccari e Giraffa, 1996) observaram que este tipo de programa tinha uma estrutura de transmissão do conhecimento de caráter seqüencial, previamente determinada, o que impedia uma adaptação individual ao estilo de cada aluno. Dadas estas

limitações, buscou-se na Psicologia Cognitiva idéias de como tornar o programa educacional mais individualizado às necessidades cognitivas de um aluno. Na própria década de 70 surgiram então os primeiros Sistemas Tutores Inteligentes (ITS- Intelligent Tutoring Systems), que passaram a usar a Inteligência Artificial. Os ITS são conhecidos também por ICAI (Intelligent Computer Assisted Instruction), ou seja podem ser vistos como um braço da evolução ocorrida com o CAI. As primeiras pesquisas em ITS focaram-se na representação do conhecimento a ser transmitido. Para tal foram usadas técnicas de Inteligência Artificial que permitiam a apresentação do conteúdo a ser ministrado de forma independente às técnicas didáticas. No entanto, a implementação de tutores baseados em computador mostrou ser um tanto quanto difícil; devido a sua complexidade, estes programas geralmente não funcionam em microcomputadores, tendo seu uso restrito aos centros universitários os quais possuem recursos computacionais para tal. Recentemente, no final da década de 90, com popularização da Internet já se dispõe de ITS na WWW (ELM-ART In: Brusilovsky, 1996). Este campo vem se beneficiando de muitas áreas da Inteligência Artificial como a da linguagem natural, dos métodos de inferência, da integração simbólica, dos diagnósticos médicos e das prova de teoremas.

Na década de 80, com a popularização dos microcomputadores, concomitantemente às pesquisas feitas em ITS, ocorreu a disseminação do CAI nas escolas. Houve uma enorme produção de material destinado a cobrir todas as áreas ensinadas, surgindo tipos distintos de softwares educacionais, baseados em CAI, como veremos a seguir: tutorial, programas de demonstração, exercício e prática, avaliação do aprendizado, jogos educacionais e simulação.

Na década de 90, houve uma tremenda evolução em termos da qualidade dos microcomputadores, especialmente dos monitores de vídeo, o que propiciou um avanço na qualidade gráfica dos softwares. Os conceitos de ‘hipertexto’- forma não linear de visualizar a informação e de ‘multimídia’ - integração de mídias resultando num documento composto, conforme vimos no capítulo 2, foram incorporados aos softwares educacionais.

Estes novos recursos levaram a um repensar nos usos e possibilidades de integração do computador, resultando na conscientização de que o computador criado inicialmente com o intuito de realizar operações numéricas, adquiria agora um novo papel, predominante ao

anterior: o de ser um veículo de comunicação. Os softwares educacionais foram amplamente beneficiados por estas descobertas.

Com o surgimento da multimídia e do hipertexto, foram negligenciadas as contribuições advindas da Inteligência Artificial e muitos programas foram criados sem seus recursos. Atualmente no entanto, já se sabe que uma interação ajustada às necessidades do aprendiz requer o conhecimento advindo das pesquisas feitas em Inteligência Artificial (Self, 1995). Mais recentemente a Realidade Virtual vem ocupando lugar de destaque em softwares de entretenimento e educacionais.

3.1.3 CAI x ITS

Vimos ao longo da evolução dos softwares educacionais que o surgimento do ITS não impediu que o CAI prosseguisse em sua trajetória. As duas concepções convivem atualmente. Afim de salientarmos as diferenças e pontos em comum entre os softwares educacionais CAI e ITS, apresentamos a Tabela 1.

Aspecto	CAI	ITS
Origem	Educação	Ciência da Computação
Bases Teóricas	Skinner (Behaviorista)	Psicologia Cognitivista
Estruturação e Funções	Uma única estrutura algorítmicamente pré-definida, onde o aluno não influi na seqüenciação.	Estrutura subdividida em módulos cuja seqüenciação se dá em função das resposta do aluno.
Estruturação do Conhecimento	Algorítmica	Heurística
Modelagem do Aluno	Avaliam a última resposta. ¹	Tentam avaliar todas as respostas do aluno durante a interação.
Modalidades	Tutorial, exercício e prática, simulação e jogos educativos	Socrático ² , ambiente interativo ³ diálogo bidirecional ⁴ e guia ⁵ .

Tabela 1 - CAI x ITS (Viccari e Girafla, 1996, pg. 13)

¹ As ações do programa estão baseadas na última resposta do aluno, considerando-se apenas V (verdadeira) ou F (falsa). Não são tratadas as dúvidas e incertezas.

² Pretendem proporcionar instrução através de diálogo com os alunos e detectam erros.

³ Trabalham um determinado assunto através de da interação com o aluno, utilizando a abordagem de diagnóstico desse aluno.

⁴ Utilizam um ambiente interativo com diálogo entre o aluno e o tutor, podendo haver alterações no comportamento do tutor face as respostas do usuário (aluno).

⁵ Alguns ITS funcionam como guias do aluno e não se utilizam de um diálogo e sim apresentam o conteúdo de outra forma.

Na prática, não há uma divisão clara entre os paradigmas: o tradicional CAI e o ITS baseado em Inteligência Artificial. Os programas atuais, em geral, não são nem totalmente fixos pré-programados, nem totalmente autônomos.

No entanto estes não são os únicos tipos de software educacional encontrados, Galvis (1997) classifica os softwares educacionais atuais de acordo com suas funções específicas:

- *demo*: o aprendiz pode controlar o ritmo e a seqüência do percurso, que visa esclarecer um determinado assunto,
- *exercitador*: procura reforçar e generalizar as habilidades através de exercícios variados que são corrigidos pelo programa com motivadores e reforçadores que ajudam o aluno a alcançar sua meta. Não cabe ao exercitador a transmissão do conteúdo, o aluno deve tê-lo aprendido por outro meio,
- *tutorial*: além das características do exercitador, o tutorial ajuda na aquisição do conhecimento através da apresentação dosada do conteúdo, como preâmbulo ou como complementação do processo de exercitação,
- *heurístico*: apoiam a descoberta e a construção dos conceitos e habilidades, a partir da atividade de busca e exploração do aprendiz na solução de problemas. Neste tipo de software é dada muita importância aos desafios relevantes, os quais despertam a curiosidade e o desejo de aprender. Difere dos tipos anteriores que são de caráter algorítmico.
- *simulador ou sistema esperto*: semelhantes aos jogos, ensinam através da vivência lúdica, da experiência e da reflexão sobre estas, as qualidades que se quer transmitir dentro do ambiente simulado que é criado pelo programa.

3.1.4 As Linhas de Pesquisa

As direções que as linhas de pesquisa têm tomado são:

- produção de material para curso através das novas facilidades computacionais;
- criação ou refinamento de ferramentas para criação de software educativo de autoria, envolvendo recursos gráficos sofisticados;

- métodos específicos de Engenharia de Software vem sendo estudados, numa tentativa de melhorar o processo de transmissão das decisões pedagógicas, feitas por especialistas em ensino para os sistemas de ensino baseados em computador . A finalidade é obter um modelo que permita a especialistas compor interações educacionais através da representação de suas decisões na forma de programas.

Para os pesquisadores de Inteligência Artificial, a finalidade principal é capturar o conhecimento adquirido por especialistas em Educação (professores) capaz de compor uma situação educacional. Ao invés das decisões resultantes do conhecimento de ensinar, é o próprio conhecimento que é representado a fim de poder ser manipulado por sistemas baseados em computador. Ou seja, transferir a especialidade, ao invés das decisões que os especialistas tomariam em determinadas situações, cria a possibilidade de o sistema tomar decisões não previstas por estes especialistas. O objetivo é tornar os programas capazes de interagir dinamicamente, tomando decisões em sua base de conhecimento.

Como o desenvolvimento de software educativo é um problema complexo, ele vem sendo tratado em partes:

- uma das partes compreende a descrição independente do resto do programa do domínio do conhecimento;
- um grande desafio é a descrição da estratégia pedagógica;
- a tecnologia de hipertexto vem sendo estudada para melhor adequação da apresentação do material didático, a interface do sistema;
- outra parte estuda como adaptar diálogos dinâmicos com o aluno;
- a Inteligência Artificial vem estudando outro ponto básico: como modelar a estrutura cognitiva do aluno.

A comunicação existente entre o software e o aluno, é fator relevante ao sucesso do mesmo, sendo para Galvis, o terceiro eixo que compõe a natureza interdisciplinar dos softwares educacionais, completando os outros eixos, que são o educacional e o informático. Para a construção de um software educativo de qualidade, deve-se buscar alcançar uma interface amigável. A ênfase deve ser mudada de ‘o que mostrar’ para ‘como apresentar’,

tendo como alvo o usuário e sua satisfação em usar o software. Surge assim na década de 90, uma nova disciplina computacional: a Interação Humano-Computador (IHC) (Rocha e Baranaukas, 2000). O IHC, como vimos no capítulo 2, procura identificar um modelo que represente o funcionamento humano, estudando os tempos de resposta perceptíveis aos estímulos visuais, auditivos e motores, bem como o funcionamento das memórias de curta e longa duração. A partir destes dados é possível definir critérios de usabilidade, ou seja heurísticas capazes de nortear a criação de programas agradáveis de usar.

As linhas de pesquisa estão sempre a frente do tempo atual. Vejamos então o que vem efetivamente sendo usado nas escolas e universidades.

3.1.5 Novos Rumos do Software Educacional

Atualmente existem três novas formas de aprendizado nas Universidades (English e Yazdani, 1997):

- Aprendizagem Auxiliada por Computador (CAL- Computer Assisted Learning)
- Aprendizagem Cooperativa
- Universidade Virtual

A primeira forma, Aprendizagem Auxiliada por Computador, constitui em geral numa modernização do uso de programas CAI. Estes programas estão agora sendo convertidos para Internet num formato novo que inclui multimídia e hipertexto, porém sua estrutura básica é a mesma das últimas duas décadas.

A segunda nova forma, Aprendizagem Cooperativa, baseia-se nas idéias de Vygotsky, (In: English e Yazdani, 1997) que podem ser parcialmente resumidas em: *o indivíduo ao falar sobre um assunto consolida seu conhecimento sobre o tema, podendo até chegar a descoberta de uma solução*. Tais idéias levaram o software educacional a um estágio de desenvolvimento mais avançado, onde a interação tradicional existente nos softwares educacionais: tutor x aluno foi ampliada para atender vários alunos simultaneamente, de forma análoga a uma aula presencial onde as discussões não se restringem entre professor e aluno, mas ocorrem entre dois ou vários alunos.

Como vimos no capítulo. 2, Piaget afirma que esta interação entre alunos leva o aprendiz a romper seu egocentrismo intelectual, percebendo a forma de pensar e a argumentação usada por seus iguais.

No X Simpósio Brasileiro de Informática na Educação, ocorrido em Curitiba, 1999, um grande número de softwares educacionais apresentou esta tendência. O software educacional cooperativo apresenta características que proporcionam ao aluno um ambiente rico em interatividade e cooperação. Ao pesquisar sobre um determinado assunto, o aluno pode incluir numa base de dados sua pesquisa, deixando-a disponível para os demais, bem como registrar suas descobertas, ou responder a dúvidas deixadas por seus colegas. Neste ambiente, os alunos trabalham no computador individualmente, lado a lado no laboratório, como ocorria anteriormente, mas dispõe de recursos computacionais que permitem a colaboração entre alunos no estudo. Em nossa Faculdade, o IME, pode-se dizer que as listas de discussão de uma disciplina na Internet, constituem um jeito simplificado de aprendizagem colaborativa.

E a terceira forma, a Universidade Virtual, proporciona uma oportunidade de aprendizado à distância. É uma nova forma na relação de ensino-aprendizagem, onde os cursos oferecidos virtualmente lançam mão dos recursos advindos das tecnologias de comunicação e da Internet, cada vez mais interativa. Além dos alunos tradicionais, as pessoas que precisam trabalhar para se manter ou a sua família, têm agora um meio de voltar a estudar, seja por prazer ou como investimento na carreira profissional.

O tutor nestes cursos, é realmente um professor, uma pessoa, ou seja a interação com o aluno tende a ser mais próxima à que ocorre numa aula presencial tradicional - aula em que professor e aluno estão presentes fisicamente no mesmo local. Nos cursos à distância há a possibilidade de ocorrerem aulas presenciais através do uso da vídeo-conferência. Ganha-se porém em disponibilidade de tempo, pois a comunicação existente entre as partes, em geral, é assíncrona, ou seja a troca de informações entre aluno e professor ocorre em tempos distintos (como ocorre na comunicação por e-mail). Na aula presencial, a comunicação é síncrona (assim como no telefone).

Os cursos em geral possuem uma estrutura semi-aberta, onde há uma seqüência pré-estabelecida de etapas que devem ser seguidas pelo aluno, intercalada por apartes do professor que coordena o progresso de cada aluno, enviando textos extra para leitura e comentando seus trabalhos.

A estrutura deste curso pode prever ou não que o software educacional corrija ou avalie o trabalho do aluno, no entanto o professor é um elemento fundamental. O software serve como um ambiente facilitador da comunicação não presencial, interagindo tanto com o professor quanto com os alunos, e é capaz de armazenar um histórico individualizado de cada aluno, além de proporcionar aos alunos de um mesmo curso a possibilidade de discutir e compartilhar uma base de informações comum.

3.2 Sistemas Tutores Inteligentes (ITS - Intelligent Tutoring Systems)

Um destaque especial é dado aos ITS neste estudo, pois representam o que de mais atual podemos buscar como modelo de software educativo, incorporando muitas das teorias educacionais vistas neste estudo. São apresentados aqui dois ITS de comportamentos e fins distintos, e, ao final, é feita uma breve análise sob alguns dos aspectos mais relevantes.

3.2.1 Application Program Tutor – APT ¹

Este tutor foi desenvolvido pela Delta Performance Systems, com fins comerciais, mas autoriza o download de um software de demonstração. É um sistema de autoria, que permite aos especialistas em computação a geração de material profissional de treinamento, em programas aplicativos para computador, que rodem em microcomputadores PC. O APT destina-se ao treinamento de usuários finais.

Possui em sua Base de Conhecimento dois modelos: o procedural e o declarativo. O conhecimento procedural ensina os procedimentos da operação, p.ex. qual tecla pressionar, onde clicar, etc.; no entanto, é limitado e não possui conhecimento da aplicação. Para este fim o conhecimento declarativo é usado. O conhecimento procedural se dá na forma de

¹ www.dpsnet.com/apt/index.html

Figura 1 - Modelo Procedural

específica.

Figura 2 - Tela do curso APT

algumas tarefas. Se acertar, passa para frente, errando, o tutor concede uma ou duas novas oportunidades ao aluno de refazer a tarefa. Persistindo o erro, o APT retorna às telas explanatórias referentes à dificuldade em questão. Sempre procedendo desta maneira.

Se considerarmos como ensino individualizado aquele onde o tutor dedica-se exclusivamente a um aluno por vez, este ITS se enquadra; mas, se considerarmos como ensino

demonstrações de tarefas e subtarefas. O conhecimento declarativo ocorre na forma de ‘palavras-chave’ – conceitos importantes que precisam ser aprendidos.

O APT captura telas da aplicação, alvo do ensino, a fim de simulá-las (conhecimento procedural). O conhecimento declarativo é um diagrama esquemático contendo os objetos e as ações envolvidas na tarefa

O Software de demonstração consiste em um exemplo para treinamento em Word da Microsoft. Ensina como imprimir um documento. Consiste em algumas telas explanatórias das ações a serem tomadas, passando para a parte prática onde a tela capturada emula a aplicação. Pedese ao aluno que realize passo-a-passo

individualizado um ensino onde o tutor procura adequar a melhor técnica pedagógica a um determinado perfil de aluno, o APT parece não possuir habilidades para tal, pois age sempre da mesma forma. No entanto, é sem dúvida uma ferramenta importante de geração de cursos de treinamento.

Neste ITS, atuei somente no papel de aluno, ou seja, não cheguei a analisar o desempenho deste software como programa de autoria.

3.2.2 Andes ²

Desenvolvido pelo Prof. Kurt VanLehn (1996), da Universidade de Pittsburgh. Foi-me permitido o download do software completo, que roda em ambiente Windows. Este ITS dedica-se ao ensino da Física clássica, newtoniana. É um projeto em andamento, iniciado em setembro de 1995 e conta com a parceria da Academia Naval Americana. Baseado nas últimas pesquisas em Ciências Cognitivas, conta com um grupo de professores de Física experientes em reforma didática para fornecerem as entradas.

O Andes foi concebido com a intenção de ser usado pelos estudantes como sua lição de casa. Sendo que as demais atividades didáticas do curso como aulas explanativas, leituras e laboratórios devem prosseguir como antes. As razões consideradas para esta escolha foram que ITS que apenas ajudam na lição parecem ser mais difundidos e apreciados do que ITS que pretendem cuidar inteiramente da instrução; a maioria dos professores espera uma dedicação de 2hs para cada hora de aula dada, das quais a maioria corresponde à resolução de problemas; mas, embora bons professores possam propiciar boas aulas, leituras e laboratório, há pouco o que fazer quanto à lição de casa.

Seus componentes são:

- *Editor de preparação de lição de casa*: este programa é usado pelos professores para a criação de tarefas de lição de casa;
- *Tutor*: este programa é usado pelos estudantes para fazer sua lição de casa.

² <http://www.pitt.edu/~gertner/andesins.html>

Sua interface gráfica é rica em subjanelas e botões, incluindo ferramentas para desenho de diagramas esquemáticos com vetores, declaração de variáveis e uma ferramenta que calcula equações escalares. Possui vários exercícios propostos, bem como exercícios resolvidos.

O Andes, ao contrário do APT possui uma interface aberta permitindo ao aluno livre escolha na solução do problema. Senti no entanto, uma dificuldade inicial em usar o Software, por sua complexidade, apesar de possuir uma ajuda on-line detalhada e bem feita.

Enquanto o APT propicia um rápido e fácil começo de uso, o Andes necessita de uma maior dedicação à aprendizagem do funcionamento do sistema, que, por não ser dirigido, põe as rédeas das ações a serem tomadas nas mãos do aluno, que, numa fase inicial, fica imóvel,

Figura 3 – Exemplo de um problema proposto pelo Andes.

sem ação, pois a rápida exploração por tentativa e erro neste sistema não nos leva adiante. Ocorre então este dilema: sem dúvida é melhor que o aluno comande o aprendizado, mas será este preço alto de mais, levando-o a desistir de usar o Software ou ironicamente a ter que fazer um curso com um tutor humano de como usar o tutor artificial?

3.2.3 *Algumas Conclusões relativas aos ITS*

Mesmo com esta pequena amostra de ITS é fácil percebermos o grande leque de idéias existente quando nos referimos ao termo 'ITS'. A esquematização das vertentes existentes procura mostrar as diferenças que coexistem:

1) Quanto à função encontramos as seguintes propostas:

- Substituição completa do tutor humano (conteúdo + exercícios).
- Substituição parcial do tutor humano (exercícios).

Vimos que a proposta de substituição total do professor não é realizada de forma eficiente.

2) Quanto ao carácter pedagógico, pode-se dizer que existem as seguintes abordagens:

- Ambiente Interativo: permite ao aluno conduzir sua aprendizagem, possui uma interface interativa, sendo o software orientado a eventos.
- Guia: conduz o aluno na aprendizagem, levando-o por uma seqüência de tópicos pré-determinados ou determinados a partir das respostas durante a interação.
- Socrático ou por Diálogo: ensina através de diálogos entre o aluno e o tutor (não foi obtido protótipo deste tipo de tutor).

3) Quanto ao tipo de utilização

- Escolar: destinado ao aprendizado.
- Empresarial: treinamento em uma determinada função, que utilize ou não o computador.

As empresas estão acreditando e investindo mais em educação à distância. Em 2000, a Cisco, no Brasil, possuía cerca de 150 cursos de treinamento feitos por software educativo,

que vão desde a apresentação da estrutura da firma aos novos funcionários até curso técnicos especializados.

Já nas escolas a utilização ainda é restrita. Um dos fatores talvez seja o baixo valor da remuneração dos professores que dificulta justificar a relação custo-benefício na implantação de softwares educativos mais eficientes. Outro fator importante é o investimento em laboratórios de informática, oneroso em sua manutenção e constante renovação. Somente nos últimos 5 anos, as escolas passaram a considerá-lo seriamente e não apenas como chamativo à matrícula de novos alunos. As empresas, por sua vez, usam os computadores em seu dia-a-dia, os cursos informatizados correspondem a mais uma das atividades feitas pelo computador, o que não ocorre nas escolas.

4) Quanto ao desenvolvimento:

- Software de autoria na geração de ITS.
- ITS destinado a um domínio específico.

5) Quanto à plataforma

- Local
- Via Internet

A plataforma via Internet vem se consagrando nos últimos três anos. É o meio mais ágil de manter as revisões do programa atualizadas, e de permitir que aja a colaboração entre os alunos envolvidos.

A enorme variedade de tipos é um dos fatores que contribuem para que os ITS permaneçam num estágio de pré-paradigma, não existe um modelo aceito pela comunidade científica que atenda as diversas facetas do que venha a ser um ITS.

O momento atual e o ITS

Vemos que infelizmente após tantos anos os ITS ainda estão em estágio embrionário, não sendo efetivamente usados nas Universidades ou Escolas, tanto que nem foram citados neste artigo recente de Yazdani (1997), que desde do início da década de 80 pesquisa sobre o assunto. O pouco sucesso dos ITS talvez esteja relacionado à alguns fatores, observados por mim, neste estudo:

- *dificuldades computacionais relativas ao tempo de resposta das linguagens declarativas:* já que uma das características essenciais do software educacional é o pronto feedback dado ao aluno, o PROUST (visto no item 3.2.1) leva cerca de 3 minutos para dar a resposta e com certeza esta não é uma resposta em tempo-real;
- *dificuldades computacionais relativas à comunicação humano-computador:* o computador possui um padrão rígido em seu protocolo de comunicação. O aluno ao ter uma dúvida, muitas vezes precisa da ajuda do professor até para expressá-la, pois como vimos em Vigotsky o aluno ao falar sobre o assunto organiza suas idéias. Sendo assim, temos de um lado um aluno confuso, e do outro uma máquina de interface limitada, não constituindo um ambiente de aprendizagem adequado. Os progressos nas pesquisas de Inteligência Artificial voltadas à compreensão da linguagem natural irão sem dúvida minimizar este tipo de dificuldade, bem como o desenvolvimento da IHC;
- *dificuldades computacionais em compor e atualizar a base de conhecimento sobre o assunto estudado:* Como vimos no capítulo 2 a construção do conhecimento baseia-se no conhecimento pré-adquirido pelo sujeito, que expande suas redes neurais ao tentar entender um novo assunto. No entanto cada aluno possui em sua mente uma base de conhecimento única, com predisposição ou facilidades distintas. Quais serão os fios que irão ser ativados no processo de conectar o novo assunto à base de conhecimento existente? Varia de sujeito a sujeito. Como então construir uma base de conhecimento restrita ao domínio estudado, sem que seja possível estabelecer ligações com conceitos conhecidos, sendo que estes conceitos variam de sujeito para sujeito? Esta base de conhecimento a rigor deveria conter em seu domínio todo o conhecimento da humanidade para poder ser flexível o bastante e atender aos diversos alunos.

Voltemos agora nossa atenção ao ensino da programação. Quais foram os melhores softwares educacionais desenvolvidos nesta área? Atenderam satisfatoriamente ao problema?

3.3 Sistemas de apoio ao ensino da programação para iniciantes

No ensino de uma linguagem de programação a novatos, as dificuldades de aprendizagem são grandes, como foi ressaltado no capítulo 1. Neste item iremos fazer uma retrospectiva dos softwares educacionais mais significativos nesta área.

Segundo Yazdani (1986), o domínio da programação de computadores é bem estruturado e conhecido pelos desenvolvedores, o que implica numa facilidade maior no desenvolvimento de um ITS (Sistema Tutor Inteligente). Citando inclusive que os únicos dois ITS à venda são nesta área: um para LISP (Lisp Tutor) e outro para Pascal (PROUST).

O ensino da programação exige um alto grau de individualização, pois o mesmo programa pode ser resolvido por uma grande variedade de modos. Em uma classe com 50 alunos, podem ocorrer 50 programas distintos para o mesmo enunciado. Esta é uma das grandes dificuldades do tutor inteligente, ou seja, descobrir quando um desvio de rota pode levar ao fracasso do programa ou este desvio é apenas um jeito alternativo de chegar ao resultado.

A seguir sete programas educativos são vistos em detalhe.

3.3.1 PROUST (Program Understanding for Student)

PROUST (Johnson e Soloway, 1985, in Yazdani, 1986) é um ITS dedicado ao ensino da linguagem de programação Pascal. Possui um analisador de ‘possíveis erros’ cometidos com maior frequência pelos iniciantes; projetado para descobrir ‘qualquer’ erro, possui uma alta taxa de acerto na identificação de erros: 70%. Uma vez descoberto o erro, o PROUST indica como este pode ser corrigido e sugere as possíveis causas que levaram ao erro. O PROUST é um sistema completo: propõe exercícios, analisa o desenvolvimento do programa feito pelo aluno e fornece sugestões construtivas.

Para poder atuar, é necessário que cada problema proposto seja previamente codificado numa linguagem específica de resolução de problemas do PROUST e posto na biblioteca de programas. Além disto, existe uma base de conhecimento dos erros mais freqüentes em linguagem Pascal, denominada biblioteca de erros comuns.

O PROUST sintetiza cada programa comparando-o com a descrição do problema em sua biblioteca de programas e fazendo hipóteses sobre como o aluno irá resolver o próximo passo

do problema. Se uma das hipóteses se ajusta à solução dada pelo aprendiz, o PROUST conclui que esta está correta. Caso contrário, procura na biblioteca de erros comuns algum que case ao código do aluno.

Vista em detalhes (Sack, 1992), a base de conhecimento do PROUST contém gols, planos e regras de erros. O modelo do aluno contém dois conjuntos de regras: as corretas e as incorretas (ou mal-formuladas). O funcionamento do PROUST consiste em identificar cada instrução do aluno com um plano da base de conhecimento. Caso não consiga fazer a correspondência, a execução do programa é suspensa e as regras de erros são usadas numa tentativa de explicar o erro. Vale-se então da 'árvore de interpretação', a qual contém os estados anteriores, e fazendo um backtracking na árvore junto com as regras de erros o programa pode analisar o erro do aluno. O ponto fraco é que as regras de erro modificam partes vitais da informação contida nos estados anteriores da 'árvore de interpretação', sendo necessária a interrupção do programa.

O PROUST foi testado em um grande número de alunos de graduação da Yale University, obtendo as seguintes médias:

- em 70% dos testes foram detectados todos os erros do programa;
- em 17% dos testes foram detectados parcialmente os erros;
- em 4% dos testes não foi possível detectar corretamente os erros, ocorrendo uma deterioração drástica em sua habilidade de reconhecimento de erros.

O ITS PROUST levou quatro anos para ser desenvolvido, a um custo de meio milhão de dólares, pela Yale University, com uma equipe de dois programadores assistentes. Foi desenvolvido em LISP, resultando em 15.000 linhas de código (4MBytes de memória). A análise de um programa leva de 3 a 5 minutos, rodando em um DEC VAX750.

Existe uma versão para computadores pessoais compatíveis com o IBM-PC: Micro-Proust, escrito em Golden Common LISP. O tempo para desenvolvimento foi de dois meses por um único programador. Possui apenas 1/5 do catálogo de erros do PROUST e leva apenas 90 segundos para dar a resposta, mas sua taxa de acerto é desconhecida.

3.3.2 Program Critic

O Program Critic desenvolvido por Sack (1992) possui a mesma intencionalidade do PROUST, ou seja, é um sistema que analisa e critica programas escritos em Pascal, feitos por alunos de computação. Sack alega que as 15.000 linhas em LISP do PROUST podem ser equivalentes a 100 linhas em PROLOG. Sua justificativa é que o excedente de código LISP está justamente na tentativa de implementar a funcionalidade de um interpretador PROLOG.

Sack alternando a linguagem para PROLOG beneficia-se do backtracking próprio e eficiente desta linguagem, evitando que ao se detectar um erro seja necessária a interrupção do programa. Sua proposta é eliminar as regras de erro da arquitetura do sistema, substituindo-as por planos de erro e gols alternativos escritos na mesma notação dos planos e gols corretos, evitando desta forma que a execução do programa precise ser suspensa.

3.3.3 Tutor LISP

Os ITS estão numa fase inicial de desenvolvimento, o que não permite que seja estabelecida uma arquitetura geral única na construção de um sistema computacional. No entanto, o desenvolvimento do Tutor Lisp propiciou definir alguns princípios que podem ser usados na construção de ITS em geral. O Tutor Lisp foi desenvolvido por John Anderson (Anderson e Reiser, 1985, in Yazdani, 1986) e seus colegas na Carnegie-Mellon University, na primeira metade da década de 80.

Os estudos prévios de Anderson sobre a Teoria do Controle de Pensamento Adaptativo (ACT- Adaptive Control of Thought Theory) levaram a construção de uma nova arquitetura de ITS, composta por quatro componentes:

- *Conhecimento do domínio*: módulo capaz de resolver problemas no domínio específico, é também referido como ‘Modelo do aluno ideal’.
- *Catálogo de Erros*: é uma biblioteca de erros e conceitos mal entendidos sobre o domínio.
- *Conhecimento pedagógico*: módulo com as estratégias de ensino do domínio
- *Interface com o usuário*: este módulo administra a interação entre o tutor e o aluno.

O módulo de conhecimento do domínio permite ao tutor resolver por si só o problema no qual o aluno está trabalhando.

A interface com o usuário constitui-se num bom editor de código fonte, que permanece quieto até que o aluno cometa um erro em seu programa, entrando então em ação através do diagnóstico do erro e apresentação de dicas de como solucionar o problema.

O Tutor Lisp contém aproximadamente 325 regras de produção de como planejar e codificar em Lisp e 475 regras para tratamento de erros. Seu grau de acerto está entre 45 e 80%.

3.3.4 ELM-Art³

Desenvolvido pelo Prof. Peter Brusilovsky (1996) da Universidade Carnegie Mellon, em Pittsburgh. É um curso completo da linguagem de programação LISP, tendo recebido prêmio de melhor software do ano. Este ITS é on-line, via Internet, ou seja não é necessário fazer download, evitando assim os problemas de instalação. Basta acessar a página do curso e inscrever-se.

Figura 4 - ELM-ART – Exercício

Figura 5 – ELM-ART Tela de Ajuda

³ <http://www.contrib.andrew.cmu.edu/~plb/home.html>

É um programa muito fácil de se usar, em forma de testes de múltipla escolha. Neste ITS há um programa a cumprir, organizado em 6 lições, subdivididas, que conduzem o aluno. No início de cada subitem há um breve texto explanativo com exemplos, seguido pelos testes. Em cada página são dois ou três testes. Depois de completá-los, as respostas são enviadas e o estudante recebe imediatamente a correção.

O conteúdo do conceito a ser transmitido é assimilado através do feedback obtido na correção imediata. A qualquer momento, este pode realizar testes que lhe permitirão pular lições de conteúdos já adquiridos. O ensino individualizado neste ITS fica mais visível, no caso de acerto dos testes, com a lição rapidamente chegando ao final, enquanto que se errarmos alguns testes, o ELM propõe novos testes, sempre diferentes dos anteriores, o que reforça o conceito que fora mal assimilado.

O ELM possui ainda uma sala de conversação entre os alunos, e uma linha direta, via e-mail, com os tutores humanos. É permitido ao aluno ver seu desempenho, e alterar o modelo do aluno, porém o programa não mostra quais são os modelos disponíveis.

Veremos agora o que vem sendo feito em nosso meio acadêmico, aqui no Brasil.

3.3.5 Sistema Tutor Asimov – UFPR

Este sistema foi apresentado por Binder e Direne (1999) no X SBIE , realizado em Curitiba. Sua implementação não está concluída, mas podemos examinar sua proposta.

O principal objetivo é apoiar o ensino de lógica de programação de linguagens imperativas/procedimentais, levando em conta os aspectos de exploração livre do ambiente de programação, diagnóstico automático de programas, visualização de células de memória, execução compassada, configuração parcial do tutoramento pelo aluno, inclusão de novos problemas e mudança da linguagem alvo do sistema.

Neste software educacional, os autores acreditam que a rigidez sintática deve ser abrandada num primeiro contato com a linguagem, concentrando os esforços na solução do problema. Deste modo é usado pseudo-código como linguagem.

O diagnóstico automático do programa, ponto central proposto, constitui na representação do problema através de uma *rede semântica*, denominada GRADE, que constitui um grafo que representa a solução de referência do problema, levando em conta as diversas possibilidades de solução. O GRADE é composto por quatro elementos básicos: nós, arcos de dependência, arcos de alternativa e arcos de continuidade. Vejamos o funcionamento deste elementos:

- *os nós*, possuem atributos que identificam a linha referida no programa fonte;
- *os arcos de dependência*, indicam que o nó destino é dependente do nó origem, ou seja o comando do nó destino só é válido se estiver depois do comando do nó origem;
- *os arcos de alternativa*, indicam que o nó destino é uma alternativa ao nó origem, em caso de falha na comparação do comando com o nó origem;
- *os arcos de continuidade*, usados para representar a ordem de continuação para alternativas.

A comparação entre a solução do aluno e a solução de referência é realizada através de um algoritmo de casamento de padrões, já conhecido pela Inteligência Artificial. No caso de não se obter um encaixe, um erro inter-comando é identificado. Uma mensagem é enviada ao aluno, baseada nas classes de erros mais comuns em programas de alunos iniciantes.

As classes de erro podem ser de dois tipos: inter-comandos e intra-comandos, mas o Asimov só trata erros inter-comandos em sua versão inicial.

O ponto chave deste software está na habilidade do especialista em construir um grafo GRADE da solução geral do problema, pois como foi visto ao longo do capítulo 1, existem *muitas* soluções para resolver um problema computacional.

3.3.6 Curso de Linguagem C – UFMG ⁴

Este curso foi implementado na UFMG - Universidade Federal de Minas Gerais pelo Núcleo de Ensino à Distância da Escola de Engenharia - fazendo

⁴ <http://ead1.eee.ufmg.br>

parte de um projeto apoiado pela Pró-Reitoria de Graduação da UFMG, através do programa PROGRAD97 / FUNDO-FUNDEP.

O curso é promovido pelo Grupo de Otimização e Projeto Assistido por Computador (GOPAC) do Departamento de Engenharia Elétrica da UFMG, sendo oferecido regularmente, a cada semestre, desde 1997. O curso é oferecido gratuitamente, mas não oferece certificados de conclusão, pois não foi desenvolvido ainda nenhum esquema de avaliação (apenas auto-avaliação), destinando-se a pessoas interessadas em seu aprimoramento pessoal.

Atualmente ele é coordenado pelo professor Renato Cardoso Mesquita do Depto. de Engenharia Elétrica, que conta com a colaboração de uma equipe técnica (monitores)

composta por 10 alunos do curso de pós-graduação em Engenharia Elétrica da UFMG (mestrado e doutorado).

Na edição do 1ºSem/2000, o número de participantes foi em torno de 2500. Inscrevi-me nesta edição, tendo participado do começo do curso.

Figura 6- Material do curso - o livro

Não é necessário nenhum conhecimento anterior em programação para participar. O funcionamento do curso gira em torno da lista de discussão⁵, principal forma de comunicação entre o professor e os alunos. O material do curso, denominado *o livro*, como pode ser visto na

⁵ cursoc@ead1.eee.ufmg.br

Figura 6, está disponível nas páginas do curso na Internet (pode ser feito download). Para praticar os exercícios, é necessário ter instalado um compilador C padrão ANSI. (são fornecidos locais para download gratuito).

O curso tem duração de oito semanas. O coordenador envia um e-mail com as instruções da semana, como ilustra a Figura 7, o aluno lê o livro do curso e interage com o professor, monitores e demais colegas usando a lista de discussão. Há um FAQ (Frequently Asked Questions) do curso, onde as dúvidas mais frequentes são listadas.

Este curso pode ser classificado como um software CAI, baseado nas Instruções Programadas de Skinner, onde o conteúdo a ser ministrado é dividido em módulos sequenciais e há uma auto-avaliação no final de cada módulo. A auto-avaliação pode ser on-line, testes de múltipla escolha, ou consistir num enunciado de um exercício de programação, que deve ser desenvolvido usando-se o compilador C. A correção é feita pelo próprio aluno, e suas dúvidas são lançadas na lista de discussão. Trata-se de uma versão modernizada de CAI, através do ambiente de colaboração existente (lista de discussão), possível graças ao uso do correio eletrônico, as dúvidas são compartilhadas entre os alunos e não necessariamente respondidas pelo professor. Este tipo de discussão sobre o assunto, faz o aluno interagir, liberando-o de seu egocentrismo intelectual, como vimos no capítulo 2.

A adequação do curso a um tipo específico de aluno é feita através do elemento humano: a equipe técnica, presente no sistema. A parte computacional do sistema não está habilitada a responder dúvidas do aluno, sendo bem limitada. No entanto este modelo de educação à distância, onde o elemento humano supre as dificuldades computacionais de interação, está sendo usado em larga escala, tanto em Universidades como a PUC-SP, como em cursos comerciais de línguas estrangeiras, entre outros.

Pessoal,
vamos as atividades da primeira semana de aulas. Esta é uma semana onde vamos "pegar leve" para que todos se acostumem primeiro com o esquema do curso, entendam como tudo vai se processar, tirem suas dúvidas e comecem a trabalhar.
Você terá as seguintes atividades a serem cumpridas:

1- Visitar a página principal do curso:

<http://www.cpdee.ufmg.br/cursos/C/>

verificar os links nela existentes visitando os itens:

- Informações Gerais (onde se descreve "o espírito do curso")
 - FAQ (onde temos as perguntas mais frequentes e suas respostas)
 - Programa do Curso(a partir do qual se acessa o "livro" do curso)
 - Área de Download (onde você podem buscar o "livro" zipado e informações para baixar o compilador)
- Monitores (conheça o pessoal que está dando suporte ao curso)
- E-mails enviados às listas (onde são armazenados os e-mails enviados a an_cursoc@cpdee.ufmg.br e a cursoc@cpdee.ufmg.br).
 - Cronograma de atividades

2- Estudar o material referente ao módulo 1:

No item "Cronograma de Atividades" estão especificadas as atividades a serem desenvolvidas em cada módulo. Um módulo corresponde a aproximadamente uma semana. Assim, seguindo o que está especificado para o primeiro módulo, na primeira semana, que vai de hoje (19/04) até a segunda-feira (26/04) você deverá:

Consultar o material inicial do curso

<http://www.cpdee.ufmg.br/cursos/C/>

Consultar a bibliografia

<http://www.cpdee.ufmg.br/cursos/C/aulas/cbiblio.html>

Estudar a aula 1, que consta da Introdução

<http://www.cpdee.ufmg.br/cursos/C/aulas/c100.html>

e Sumário

<http://www.cpdee.ufmg.br/cursos/C/aulas/c110.html>

(OBS.: o sumário é somente para referência. Você não precisa visitar os links constantes nele!)

Estudar a aula 2: Primeiros Passos, que apresenta uma introdução rápida a conceitos básicos da linguagem C.

<http://www.cpdee.ufmg.br/cursos/C/aulas/c200.html>

Nesta página, você deve visitar e estudar todos os links.

3- Desenvolver as atividades de avaliação para o módulo 1:

a) Fazer a auto-avaliação on-line para as aulas 1 e 2. Esta auto-avaliação pode ser acessada a partir da página

<http://www.cpdee.ufmg.br/cursos/C/aulas/c280.html>

As instruções para a auto-avaliação estão presentes na própria página.

b) Fazer os exercícios de auto-avaliação presentes nas páginas

<http://www.cpdee.ufmg.br/cursos/C/aulas/c220.html>

<http://www.cpdee.ufmg.br/cursos/C/aulas/c230.html>

<http://www.cpdee.ufmg.br/cursos/C/aulas/c240.html>

<http://www.cpdee.ufmg.br/cursos/C/aulas/c250.html>

<http://www.cpdee.ufmg.br/cursos/C/aulas/c260.html>

Estas são as atividades previstas para esta semana. Se você tiver dificuldades, dúvidas, etc., tente ler com atenção o material, consulte o FAQ e, caso a dúvida persista, escreva para a lista de discussão do curso: cursoc@cpdee.ufmg.br. Alguém irá auxiliá-lo!

Bom trabalho para todos,

Renato.

| \ Renato Cardoso Mesquita | \ ____oo_

renato@cpdee.ufmg.br===== (_ | / ____>

r.mesquita@ieee.org

Departamento de Engenharia Elétrica | // Fax: +55 31 499-4810 Universidade Federal de Minas Gerais

| // Fone: +55 31 499-4838 Av. Antônio Carlos 6627, Pampulha, Belo Horizonte, MG, 31270-901,

Brasil <http://ead1.eee.ufmg.br/~renato> <http://moo.cpdee.ufmg.br/>

Figura 7 - Comunicação inicial do curso por e-mail

Na PUC/SP, podemos citar três cursos a distância:

- Projeto NAVE: ambientes virtuais de aprendizagem colaborativa. Coordenado pelo Prof. Dr. Fernando Almeida (PUC/SP), em parceria com a IBM e o Ministério da Ciência e Tecnologia;
- Curso de Pós-graduação Lato Sensu em Desenvolvimento de Projetos Pedagógicos com Uso das Novas Tecnologias. Coordenado pelo Prof. Dr. José Armando Valente (UNICAMP - PUC/SP), em parceria com o Ministério da Educação sob o PROINFO (Programa Nacional de Informática na Educação);
- Projeto de Formação de Professores em EaD. Coordenado pela Profa. Dra. Maria Candida Moraes (PUC/SP) em parceria com a OEA (Organização dos Estados Americanos) e SME (Secretaria Municipal da Educação de São Paulo).

Vemos aqui que a necessidade de individualização do ensino continua necessária e presente, mas o caminho encontrado para viabilizar este processo foi o uso combinado de recursos computacionais e humanos, usando o que cada um tem de melhor. O computador atua como um veículo de comunicação e o professor como um oráculo do aluno.

Os recursos do hipertexto estão presentes, agilizando o acesso a informação.

Ao ser perguntado sobre sua opinião sobre o curso, o Prof. Renato Cardoso Mesquita, coordenador do curso, respondeu:

“A minha visão do curso é a de que o ponto forte dele é a interação que existe entre os alunos. Ao final de cada curso existe um sentimento de grupo bastante forte entre eles, tanto que, geralmente, eles “fundam” novas listas para continuar interagindo ao seu final. Isto é bem legal ...O ponto fraco é o mesmo! Como a interação é muito intensa, várias pessoas desistem do curso por “não agüentarem” receber tantos e-mails. Por isto, eu digo que, para fazer este curso, o perfil do aluno é de uma pessoa que gosta de se comunicar por e-mail e que gosta de auxiliar seus colegas no desempenho de suas tarefas. O curso tenta estimular a cooperação entre todos os participantes.”

Nas caixas da Figura 8 estão algumas das primeiras dúvidas enviadas para a lista de discussão. Estas dúvidas serão levadas em consideração, ao levantarmos os requisitos do sistema, no capítulo 5.

Figura 8 - Dúvidas enviadas a lista de discussão

3.3.7 Projeto MacMultimídia – IME – USP ⁶

Figura 9 - Projeto Mac Multimídia

O Projeto Mac-Multimídia desenvolvido por professores e alunos de graduação e pós-graduação do Departamento de Ciência da Computação do IME-USP , do qual participei, caracterizou-se por buscar um novo caminho de ensino para a Introdução à Computação. Sua proposta não é substituir o professor em sala de aula, mas auxiliar o aluno em seus primeiros exercícios de programação na linguagem C. Segundo a Proposta enviada a Capes, seu objetivo é o seguinte:

⁶ <http://www.ime.usp.br/~macmulti>

“ O projeto MAC-Multimídia: Material Didático Multimídia para as disciplinas MAC-11x (Introdução à Computação) tem como objetivo a geração de material de apoio ao aprendizado para as disciplinas de Introdução à Computação que utilize efetivamente os recursos tecnológicos introduzidos pela popularização dos microcomputadores e kits multimídia. Mais especificamente, propomos disponibilizar em um site da Internet e através de um CD-ROM um material didático baseado em hipertextos que complemente efetivamente o conteúdo das aulas. Acreditamos que uma apresentação que conjugue som, imagem, vídeo e animação de soluções para problemas propostos, na forma e velocidade dirigida pela interação do aluno com o sistema, facilitará o aprendizado e motivará o aluno a ampliar o conhecimento adquirido em aula. “(Proposta, 1998)

O principal ponto buscado foi o de auxiliar o processo cognitivo do aluno.

O material desenvolvido abrange três áreas distintas:

6) História da Computação

Como pode ser visto na Figura 8, mostra as origens do calculo até chegar aos computadores atuais, através de linguagem lúdica e muitas ilustrações.

Seu objetivo é mostrar ao aluno a evolução do cálculo, passo a passo, proporcionando um entendimento mais específico das funções e possibilidades de evolução dos computadores.

Figura 10 - História da Computação

Figura 11 – Lista de Exercícios

7) Listas de Exercícios

Uma relação bem completa de exercícios, cujo uso pelos alunos da disciplina já é feito há anos, através de material impresso.

Alguns dos enunciados possuem a resolução em C e Pascal.

8) Problemas resolvidos com simulação,

Alguns exercícios mais significativos foram selecionados da Lista de Exercícios. A fim de auxiliar o aluno na resolução de um problema, o software educativo propõe que sejam seguidos três passos:

1º Passo: Entendimento do Problema

Este passo consiste em se entender o que o programa deve fazer. Em alguns casos, envolve conhecimento prévio matemático sobre tópicos específicos como fatorial, número primo, entre outros, que devem ser compreendidos. Nesta etapa deve-se saber resolver mentalmente o problema. É fundamental que o aluno saiba claramente qual será a saída da função para uma determinada entrada, *antes de iniciar* a programação da função. O software educativo exibe através de uma animação do professor em sala de aula, a resolução do problema.

Figura 12 - Animação de um problema

Figura 13 - Simulação

2º Passo: Codificação

O aluno é levado através de perguntas interativas (testes), a ir desenvolvendo o algoritmo do problema. A cada questão do teste corresponde, em geral, a uma linha de código do programa. Deste modo pode-se ter uma noção de como ocorre a passagem da linguagem natural para a linguagem de programação.

3º Passo: Simulação

Nesta etapa o programa é simulado passo a passo. Os valores assumidos pelas variáveis são mostrados a cada interação, promovendo uma maior entendimento da lógica do programa.

3.3.8 Algumas conclusões sobre os sistemas de apoio ao ensino da programação

Vemos nestes softwares apresentados, como está deficiente o desenvolvimento de softwares educacionais, no sentido de atender às necessidades e expectativas dos usuários, se tomarmos como base de comparação outras categorias de características igualmente interativas como editores de texto, ou softwares de navegação em hipertexto - 'browsers'.

3.4 Conclusões deste capítulo

Na evolução do software educacional observa-se a importância da comunicação deste com o aluno, ponto chave da relação ensino-aprendizagem. A evolução dos softwares mostra que o alvo continua sendo o modelo do tutor humano, um tutor personalizado ao aluno com alto grau de interatividade, ideal buscado pelos ITS, os quais ainda estão em estágio de pré-paradigma. Para que suas metas venham a se concretizar, é necessária a evolução, em suas várias formas, do relacionamento do homem com o computador.

As novas disciplinas da Ciência da Computação mostram esta tendência, como o IHC que preocupam-se exclusivamente com o desenvolvimento da interatividade, visando uma comunicação mais abrangente em seus diversos níveis.

Atualmente os cursos oferecidos virtualmente lançam mão dos recursos advindos das tecnologias de comunicação e da Internet, proporcionam uma oportunidade de aprendizado à distância aos excluídos do ensino tradicional. Pode-se destacar também a importância da

aprendizagem cooperativa, baseada nas idéias de Vygotsky, presente em muitos cursos à distância.

Na área específica do ensino da programação, observa-se a carência de softwares específicos, e o alto interesse que há pelo assunto – havia 2500 inscritos no curso de C da UFMG. O Projeto MacMultimídia teve um bom retorno também.

A partir dos estudos feitos constatamos que na prática, não há uma divisão clara entre os paradigmas: o tradicional CAI e o ITS baseado em Inteligência Artificial, a proposta a ser apresentada nesta dissertação busca justamente conciliar estes dois polos: o possível e o desejável, sendo assim apesar de não serem usadas técnicas de IA, o sistema proposto quer deixar o aluno ser o guia de seu aprendizado, propondo um ambiente interativo semi-dirigido, onde as atividades são propostas mas o aluno possui diversos modos de realizá-la.

Capítulo 4

A Análise de Requisitos em Engenharia de Software

4.1 A Engenharia de Software

A Engenharia de Software surgiu devido aos problemas ocorridos no desenvolvimento do software, onde em geral os sistemas são feitos sob encomenda. A necessidade de prever o tempo e os custos, a fim de viabilizar o negócio tanto por parte do desenvolvedor, como por parte do cliente, bem como insatisfações do cliente quanto à qualidade do produto, à falta de garantia, e ao alto custo de manutenção, levou ao estudo apurado das etapas de desenvolvimento do software. Da mesma forma, o questionamento sobre quais seriam as técnicas mais adequadas a serem utilizadas. O processo de criação de software foi estudado sob o prisma da Engenharia, afim de torná-lo exeqüível. Este estudo resultou em melhorias nas áreas de qualidade, manutenção e satisfação por parte do cliente.

Uma das primeiras definições foi dada por Fritz Bauer (in Pressman, 1995, p.31):

“ O estabelecimento e uso de sólidos princípios de engenharia para que se possa obter economicamente um software que seja confiável e que funcione eficientemente em máquinas reais.”

Pressman destaca que a Engenharia de software abrange três componentes básicos:

- *Métodos*: proporcionam os detalhes de *como* construir o software. Englobam tarefas como planejamento e estimativa de projeto, análise de requisitos de software e de sistemas, projeto da estrutura de dados, arquitetura de programa e algoritmo de processamento, codificação, teste e manutenção.
- *Ferramentas*: existem para sustentar cada um dos métodos. São muito usadas ferramentas CASE.
- *Procedimentos*: constituem o elo de ligação entre métodos e ferramentas. Definem a seqüência em que os métodos são aplicados.

Segundo Brooks (1995), um programa feito por um único programador, quando pronto, está apto a funcionar para seu autor e na plataforma na qual foi desenvolvido. Para que o programa se torne mais útil, há duas maneiras:

- transformá-lo num produto de programação, podendo ser executado, testado e com a manutenção feita por qualquer um,
- transformá-lo num sistema de programação, isto é, numa coleção de programas interativos, coordenado em funções e disciplinado no formato.

Mas o objetivo principal é criar um Produto de Sistema de Informação que reúna as características do produto e do sistema de programação.

O processo de desenvolvimento de software, quando bem conduzido, é um conjunto de atividades e resultados associados que resultam num Produto de Sistema de Informação:

- 1) Especificação do software, onde sua funcionalidade bem como as restrições de operação são definidas.
- 2) Desenvolvimento do software, onde o software é produzido segundo as especificações.
- 3) Validação do software, quando o cliente assegura que o funcionamento do software atende às suas necessidades.
- 4) Evolução do software, quando o sistema precisa acompanhar as novas necessidades do cliente.

4.2 Análise de requisitos e suas etapas.

A Análise de Requisitos é a primeira atividade técnica no desenvolvimento do software, e pode ser entendida por definir os serviços que um sistema deve realizar, sua interface com os demais elementos, e sob quais restrições o sistema deve operar. Os requisitos do sistemas devem estabelecer *o que* o sistema deve fazer ao invés de *como* isto será feito. O produto final deste análise fica registrado no Documento de Requisitos de Software.

O termo *Análise de Requisitos* ou *Engenharia de Requisitos* segundo Stokes (1993) refere-se a uma coleção de processos: extração (elicitation), especificação, verificação e validação:

- 1) *A extração* de requisitos é o exercício de agrupar as informações a fim de verificar exatamente o que o cliente ou usuário está requerendo.
- 2) *A especificação* costuma referir-se tanto ao processo como ao resultado deste processo. Resulta em um documento denominado *Especificação dos Requisitos*, onde toda a informação obtida sobre o processo está reunida. É onde ocorre o maior esforço de análise, o fluxo e o conteúdo da informação são avaliados, as funções do software são definidas e detalhadas, o comportamento do software deve ser entendido no contexto do ambiente, as características da interface devem ser estabelecidas e as restrições do projeto devem ser incluídas.
- 3) *A verificação* é feita para assegurar que este documento não contém inconsistências, os requisitos não devem ser conflitantes entre si.
- 4) *A validação* preocupa-se em assegurar que o documento descreve com precisão o sistema que o cliente deseja, incluindo todas as funcionalidades e restrições impostas pelo cliente. O sistema deve ser realista, não tem sentido especificar um sistema que é irrealizável.

Sommerville (1995) adota o termo *definição de requisitos* para a fase inicial do processo, quando os requisitos estão sendo estabelecidos junto ao cliente e correspondem a macro especificações de serviços que o sistema deve realizar, correspondem a um documento em linguagem natural, acrescido de alguns diagramas; já o termo *especificação de requisitos* é usado no outro extremo do processo, quando os requisitos estão detalhadamente definidos e um documento preciso é gerado, também conhecido por *Especificação Funcional*. O Documento de Requisitos de Software une a definição dos requisitos e a especificação dos requisitos.

Na condução do processo alguns papéis estão envolvidos: os requerentes, os facilitadores e os implementadores. *Os requerentes* são os clientes e usuários, e representam as pessoas que precisam do sistema. *Os facilitadores* são analistas, e seu papel é o de desenvolver, ao longo do processo, as técnicas de extração, especificação, verificação e validação, numa descrição precisa do sistema que o requerente quer. Estando o Documento de requisitos de Software pronto, os *implementadores*: engenheiros, projetistas, e gerentes de projeto constroem o sistema.

O gerenciamento dos requisitos tem início com o processo de extração de requisitos e é aplicado em todas as atividades do ciclo de vida do software. Sua principal atividade é a rastreabilidade, a qual possibilita o acompanhamento dos requisitos desde sua formulação até a implementação do sistema, vinculando cada um dos requisitos a sua representação em cada fase do processo de software.

4.2.1 O que são requisitos ?

Os requisitos são uma coleção de sentenças que devem descrever de modo claro, sem ambigüidades, conciso e consistente todos os aspectos significativos do sistema proposto. Eles devem conter informação suficiente para permitir que os implementadores construam um sistema que irá satisfazer os requerentes, *e nada mais*.

Como propõe Sommerville(1995), um requisito é tratado como funcional, quando descreve um serviço ou função que o sistema deve realizar. Paralelamente podem haver requisitos não-funcionais, os quais são restrições impostas ao sistema (p.ex. tempo de resposta), ou ao desenvolvimento do sistema (p.ex. uso de determinada linguagem de programação).

4.2.2 Porque precisamos de requisitos ?

O principal objetivo é fornecer informações necessárias e suficientes para as demais etapas. À medida que o sistema computacional torna-se mais complexo, ocorre um aumento no número de erros. Este problema é agravado pelo fato de sistemas computacionais estarem cada vez mais presentes em aplicações onde falhas não podem ser toleradas. Por este motivo, a análise de requisitos desempenha um papel importante na prevenção de erros.

4.2.3 Como o analista obtém requisitos?

Os requisitos são obtidos através de:

- *Encontros com os requerentes* onde inicialmente são feitas perguntas mais genéricas e gradualmente são solicitados detalhes do sistema. É importante ao analista distinguir a operação atual da operação futura. Muitas vezes o requerente quer automatizar seu processo,

mas não sabe exatamente o que pode fazer um computador, ou que tipos de novos serviços podem ser feitos pelo sistema. Mas em geral, o requerente entende o processo do seu negócio. Este pode estar sendo feito com ou sem computador. Através da modelagem do sistema atual, pode responder com mais segurança se o modelo corresponde ao seu processo; paralelamente, familiariza-se com o método usado na modelagem, e quando o analista for lhe mostrar o modelo do novo sistema, este já poderá entendê-lo mais facilmente.

Figura 1 - Obtenção de modelos do sistema

- *Informações do domínio*, o analista deve pesquisar ao máximo o domínio a ser analisado. O vocabulário próprio deste deve ser aprendido e registrado num glossário. Algumas expressões ou termos comuns têm seu sentido modificado pela comunidade do domínio, devido à necessidade de distinguir objetos ou processos próximos. Uma vez habituado aos jargões da comunidade, a comunicação do analista com o requerente torna-se mais eficaz.
- *Informações de sistemas existentes*, através de contato com especialistas no domínio, são obtidas informações relevantes para o desenvolvimento do software, as quais o requerente normalmente não tem conhecimento. Experiências que deram certo/errado servem de guia na hora de tomar decisões.
- Normas e regulamentos, normalmente constituem os requisitos não funcionais, impostos por instituições do setor ou governamentais.

4.2.4 Como conduzir o processo de obtenção dos requisitos?

Não existem, no momento, técnicas capazes de lidar satisfatoriamente com todas as facetas da análise de requisitos, entretanto, há ferramentas e técnicas capazes de resolver com eficiência, parte do problema .

- *Rápida Prototipação:* trabalha com dois dos maiores problemas: validação dos requisitos e representação dos mesmos de forma compreensível aos diferentes leitores. Ataca também um dos problemas do desenvolvimento do software, pois ‘não há evidências de como o sistema está, para o requerente’ até que virtualmente o projeto esteja concluído. Mostrando aos requerentes um protótipo, eles podem clarear suas idéias, identificando exatamente o que desejam. As desvantagens são que se o sistema é grande e complexo, o protótipo resultará num grande esforço de trabalho. E, em alguns sistemas específicos, o protótipo é irrealizável. Além disso, há a desvantagem de que, como o protótipo implementa apenas parte dos requisitos, a validação por parte do requerente será consequentemente parcial.
- *Animação:* é similar à rápida prototipação, porém as especificações são simplesmente executadas, como um filme. A animação é um recurso mais pobre que a prototipação, pois demonstra apenas alguns aspectos dos requisitos.
- *Revisões:* constituem em geral a abordagem mais simples, que consiste na leitura dos requisitos. A desvantagem está no gasto excessivo de tempo, especialmente em grandes projetos.
- *Prova das propriedades do sistema:* baseia-se na crença que o uso de linguagens formais irá fornecer a resposta aos problemas da Engenharia de Software. Através de provas chega-se à conclusão de que o sistema opera corretamente. No entanto, existem muitas restrições ao uso de linguagens formais. Na prática provas lógicas são raramente usadas, a não ser em centros acadêmicos ou para pequenos sistemas. O desenvolvimento formal tem um preço, e algumas provas são inviáveis. Por exemplo, validade e completude são assuntos subjetivos que não podem ser provados, além disto, na resolução das provas ocorre uma ‘explosão exponencial’. E no caso de uma prova falhar, identificar exatamente a causa não é tarefa fácil.
- *Abordagem por base do conhecimento:* é feita através da modelagem do ambiente no qual o sistema irá operar; espera-se que o mundo real possa ser trazido para dentro da representação

abstrata. Exatamente como construir um modelo do ambiente e guardá-lo na base de conhecimento ainda é assunto de pesquisa. O modelo poderia ser uma interface estática e pré-definida entre a realidade e o formalismo. As desvantagens são que, modelar o mundo real é uma atividade que consome muito tempo, e nada nos assegura sua consistência em relação ao mundo real.

- *Separação de interesses:* refere-se aos problemas específicos dos diversos leitores. Evita forçar a digestão da leitura total a todos os envolvidos. Para tal pode-se dividir os requisitos em diferentes áreas. A dificuldade está em encontrar uma divisão adequada dos requisitos, devido à complexidade das relações.

4.2.5 O Método Volere

É um roteiro bem estruturado e completo para obtenção de requisitos. Desenvolvido por Robertson e Robertson (1999), este método abrange as diversas etapas do processo: identificação das pessoas e sistemas adjacentes envolvidos, identificação das ações e mensagens trocadas a partir de eventos, obtenção de requisitos funcionais e não-funcionais, resolução de conflitos entre requisitos, critérios de qualidade e protótipos. O resultado final da aplicação da técnica é o Documento de Requisitos de Software.

Usaremos este método, pois na ausência de um método específico para o domínio da educação, este método além de reunir de modo sistemático as várias etapas, permite que sejam incorporadas ou adaptadas suas sub-etapas.

4.2.6 O Documento de Requisitos de Software

O Documento de Requisitos de Software é lido por pessoas de distintas formações, no entanto ele deve ser claro e compreensível a todos os envolvidos no desenvolvimento do sistema: os clientes, usuários, analistas, projetistas, engenheiros e gerentes de projeto.

Esse documento deve servir como meio de comunicação aos envolvidos no desenvolvimento.

O Documento de Requisitos de Software é composto em geral pelas categorias de informação:

- 1) *Introdução*. a necessidade do sistema deve ser justificada, devendo situá-lo dentro das estratégias do negócio, bem como ele irá se relacionar com os demais sistemas.
- 2) *Glossário*. Contém a descrição dos termos técnicos usados no documento.
- 3) *Descrição do Contexto do Sistema*. Quando os implementadores estão a par do ambiente no qual o sistema irá atuar, normalmente o sistema tem mais chances de ter sucesso. As informações de contexto podem conter uma descrição do meio-ambiente, bem como sentenças sobre os objetivos principais do sistema.
- 4) *Descrição dos Requisitos Funcionais*. Um conjunto de sentenças em linguagem natural e diagramas que sejam compreensíveis pelo requerente, descrevendo a funcionalidade do sistema. Estes requisitos englobam não somente o comportamento externo que pode ser observado, mas inclui definições da funcionalidade interna que deve resultar no comportamento externo esperado.
- 5) *Definição dos Requisitos Não-Funcionais*. Em muitas aplicações, os resultados obtidos pela computação não são aplicáveis, a menos que satisfaçam uma série de restrições, que podem ser performance, tempo de resposta, número de usuários, normas de segurança, e normas de qualidade. Engloba também as restrições que os requerentes estipulam: compatibilidades do sistema, escolha do hardware, escolha do sistema operacional, entre outras. Estas restrições não alteram o funcionamento geral do sistema, mas restringem os engenheiros a subconjuntos limitados das possíveis implementações. Estas restrições devem ser incluídas no documento.
- 6) *Protocolos de dados e comunicações*. Muitos sistemas irão se comunicar com outros sistemas externos, e a forma que esta comunicação é feita precisa constar no documento, porém geralmente isto é feito em alto nível, sem entrar em detalhes.
- 7) *Especificações dos Requisitos*. Os requisitos funcionais e os não funcionais são detalhados. Esta parte do documento deve conter a análise dos requisitos, para tal são desenvolvidos modelos do sistema que mostram o relacionamento entre os diversos componentes. É aconselhável a inclusão de vários tipos de modelo, cada qual ressaltando aspectos distintos do sistema. Os requisitos devem ser analisados quanto a:

- Validação: o sistema deve atender satisfatoriamente aos diversos tipos de usuários, logo, é necessário que seja estabelecida uma concordância sobre o sistema entre os usuários.
- Consistência: os requisitos não podem ser conflitantes.
- Completude: a definição deve incluir todos os requisitos e restrições pretendidas pelos usuários do sistema.
- Realismo: não existe sentido especificar requisitos que sejam irrealizáveis, seja por falta de hardware ou falta de tecnologias de software.

8) *Gerenciamento do Projeto*. Informações sobre prazo de entrega, possíveis alterações, aspectos do ciclo-de-vida, detalhes da instalação, manuais, e informações sobre treinamento.

O Documento de Requisitos de Software é geralmente um enorme volume, escrito em linguagem natural. A ambigüidade da linguagem natural, as dificuldades envolvidas em sua verificação e validação, contribuem por torná-la inadequada sob o ponto de vista da análise. No entanto por falta de uma real alternativa, a linguagem natural é freqüentemente usada. Embora existam muitos métodos formais, seu uso em larga escala não foi testado ainda. Dos métodos formais existentes, nenhum é capaz de expressar os tipos de requisitos detalhados no Documento de Requisitos de Software.

Isto ilustra um dos mais graves problemas da análise de requisitos: a necessidade de precisão das especificações implica no uso de notação formal, a qual impossibilita a clareza e interpretação corretas por parte das pessoas envolvidas no desenvolvimento do sistema.

4.3 Problemas com os requisitos

Segundo Brooks (in Stokes, 1993, p.16/5), podemos dividir os problemas com requisitos em duas categorias: *essenciais e acidentais*. Problemas próprios dos requisitos, que ocorrem toda vez que a análise de requisitos é feita, classificam-se em *essenciais*. Estes problemas são insolúveis, embora existam técnicas que podem minimizá-los. Problemas que não são próprios dos requisitos mas que surgem em contextos específicos, são chamados de *acidentais*.

4.3.1 Problemas Essenciais

A origem destes problemas está em que os requerimentos são a interface entre o requerente e o analista. Para tal é necessário o uso de representações abstratas que descrevam o funcionamento e a interação do sistema. Através destas representações abstratas é criado um modelo do mundo real, ou da parte dele correspondente ao sistema, sendo que os requerimentos devem ser capazes de descrever sistemas grandes e complexos.

Linguagens para a Análise de Requisitos

Idealmente gostaríamos de usar uma linguagem que possuísse simples interpretação, ou seja a linguagem deve ser *isomórfica com* o mundo real, isto é, cada palavra desta linguagem deve referir-se a apenas uma *coisa* do mundo real. Este tipo de linguagem evitaria que as descrições fossem ambíguas, e permitiria que ambos, o analista e o requerente, a compreendessem corretamente. Entretanto, filósofos por muitos anos procuram desenvolver métodos de análise racional que pudessem permitir a uma pessoa raciocinar sobre o mundo diretamente. Os problemas estudados podem ser vistos na distinção feita por Kant entre o raciocínio analítico e o sintético. No raciocínio analítico, qualquer sentença pode ser reduzida a falso ou verdadeiro, podendo ser interpretada somente por este prisma. Contrariamente, no raciocínio sintético, uma proposição só pode ser julgada falsa ou verdadeira após ser feita uma observação do mundo real, do contexto no qual esta sentença está inserida. Esta observação é subjetiva, isto é, indivíduos distintos podem chegar a conclusões diferentes. Sendo assim, não podemos obter verdades absolutas, no que tange ao mundo real. Outro aspecto importante nas sentenças sintéticas é o tempo, sugerindo a inclusão de modelos do ambiente nas especificações de requisitos. Por exemplo, se quisermos saber se a sentença é verdadeira: ‘Juscelino Kubitschek é Presidente do Brasil’ precisamos definir a qual data nos referimos. A validade de qualquer requisito será dependente do tempo. Entretanto se captarmos o ambiente no qual estão os requisitos, as mudanças ocorridas no ambiente serão refletidas no modelo do ambiente, e a revalidação se dará de modo controlado. O problema essencial a ser buscado aqui é o de selecionar uma representação apropriada para a linguagem formal, que reduza a

natureza subjetiva de sua interpretação. Precisamos forçar o raciocínio sintético a ser menos subjetivo a fim de trazer as sentenças de requisitos a uma ótica (quase) analítica.

Compleitude

Implica na não existência de pontos falhos nos requisitos. A Especificação descreve os fatos relevantes, e somente estes, do sistema desejado. A completude é subjetiva, não existem limites rígidos impostos, conseqüentemente não há um término pré-definido da especificação dos requisitos. Somente o requerente pode decidir se a completude está correta, mas, infelizmente, o requerente muitas vezes não tem certeza do que quer. Devemos nos assegurar que o requerente interpretou os requisitos da forma que o analista pretendia.

Validação

É o processo no qual o requerente deve indicar se os requisitos refletem suas idéias. O requerente assegura-se de que os requisitos descrevem o sistema 'correto'. Aqui o problema é o mesmo: assegurarmo-nos de que o requerente interpretou corretamente os requisitos. Uma abordagem interessante, é o uso de modelos de ambiente. O desenvolvimento deste modelo capta os aspectos relevantes do domínio da aplicação, e o requerente pode, mais facilmente, validar os requisitos.

Tamanho e complexidade

Se o sistema é grande e complexo, seus requisitos refletirão isso. Sistemas grandes e complexos irão necessitar de requisitos complexos. No entanto existem exceções, alguns sistemas são fáceis de especificar e difíceis de implementar, enquanto que outros são fáceis de implementar e difíceis para especificar.

4.3.2 Problemas Acidentais

Estes problemas decorrem da ambigüidade existente na linguagem, descrito no item anterior:

Problemas em expressar requisitos.

Verificação

É o processo de conferir se a descrição do sistema em um nível de abstração está consistente com os demais níveis. Este processo é essencialmente analítico, havendo a possibilidade de serem desenvolvidas técnicas mecânicas que realizem a verificação.

Consistência

Revê cada sentença, assegurando-se que não existam sentenças conflitantes. Como na verificação é um processo puramente analítico, sendo que já existem sistemas para tal.

Modificabilidade

É a possibilidade de alterar os requisitos de modo controlado, sem que haja a necessidade de revalidar todos os requisitos.

4.4 Modelos do Sistema

Os analistas avaliam os problemas atuais e as informações desejadas como entrada/saída e criam modelos do sistema. De acordo com Pressman modelos são representações tanto do mundo físico como do mundo lógico, são feitos usando-se diagramas que representam graficamente o domínio, melhorando o entendimento das informações adquiridas.

Segundo Sommerville a *representação do modelo* contém toda a informação sobre o que é representado, enquanto que a *abstração do modelo* deliberadamente simplifica o domínio,

tomando deste somente o que é mais importante, sendo assim ao traduzir um livro do inglês para o português o mesmo conteúdo é representado (modelo) em outra linguagem, enquanto que ao resumir este mesmo livro, seu conteúdo é abstraído.

Figura 2 - Análise e Modelos

Os métodos que os analistas empregam para conceber o novo sistema são conhecidos como *métodos de análise ou modelagem do sistema*. Estes métodos ensinam a construir modelos abstratos do sistema.

Como salientam Booch et al. (1999), “um modelo é uma simplificação da realidade”, e “nós construímos modelos para compreender melhor o sistema que estamos desenvolvendo”. Os modelos seguem a técnica da Divisão e Conquista apresentada por Edsger Dijkstra, onde um difícil problema é quebrado em uma série de problemas menores. Além disto, através da modelagem trabalhamos em altos níveis de abstração, aumentando a capacidade do intelecto humano.

Através da modelagem atingimos quatro objetivos:

- 1) Modelos nos ajudam a visualizar um sistema como ele é ou como desejamos que ele seja.
- 2) Modelos nos permitem especificar a estrutura ou o comportamento de um sistema.
- 3) Modelos nos dão um molde que nos guia na construção de um sistema.
- 4) Modelos documentam as decisões tomadas.

4.4.1 Modelos Correntes

Existem muitas técnicas de modelagem, destacamos a seguir as mais expressivas, baseadas em diferentes tipos de abstração:

- *Casos de Uso*: uma descrição de um conjunto de seqüências de ações, resultantes da interação do sistema com um ator (um tipo de requerente). As ações produzem um resultado que pode ser observado pelo ator .
- *CORE – Controlled Requirements Expression*: o domínio a ser modelado é limitado e particionado em perspectivas (viewpoints) que representam organizações, homens, entidades de hardware ou software.
- *DFD – Diagrama de Fluxo de Dados*: concentra-se no fluxo de dados e nas transformações funcionais dos dados, deixando de lado detalhes da estrutura de dados.
- *ERAE – Entity-Relation-Attribute-Event*: uma tentativa de direcionar a definição dos requisitos do prisma “orientado-a-computador” para “orientado-a-problema”. O ambiente ganha destaque, sendo os requisitos estabelecidos em relação ao ambiente, como resultado a validação e a completude são facilitadas.

- *JSD – Jackson System Development*: concentra-se em modelos de domínio de informação do mundo real.
- *KBRA – Knowledge-Based Requirements Assistant*: é um subsistema do knowledge-based software assistant, nascido da aspiração de criar um paradigma de ciclo-de-vida baseado em conhecimento, capaz de formalizar os requisitos usados para o desenvolvimento do software.
- *MAL – Modal Action Logic*: foi definido como uma tentativa de produzir o formalismo matemático necessário à especificação de requisitos, sem que para tal fosse necessário o conhecimento matemático para escrever e entender as especificações em MAL. É baseado na lógica de 1^a. ordem.
- *Modelo de Estímulo-Resposta*: diagramas de transição de estado mostram como o sistema reage a eventos internos e externos.
- *Modelo do Processo*: mostra as principais atividades do processo.
- *PAISLey – Process-oriented, applicative, and interpretable specification language*:
- *RLP – Requirements Language Processor*: na tentativa de solucionar o problema dos diversos tipos de leitores, o RLP permite o desenvolvimento de linguagens formais para os requisitos de especificação das aplicações.
- *RML – Requirements Modelling Language*: Assim como a ERAE, procura dar destaque ao ambiente do sistema. Sua especificação consiste numa série de objetos relacionados, que procuram representar conceitos do mundo real. Os objetos são agrupados em classes, que agrupam características comuns.
- *SCS – Structured Common Sense*: foi desenvolvido como um método de auxílio ao desenvolvimento de especificações formais a partir de conceitos da aplicação. É dirigido a extração e formalização dos requisitos usando o MAL.
- *SREM – Software Requirements Engineering Methodology*: é uma ferramenta baseada na engenharia de requisitos e validação de sistemas. É composto por um tradutor para RSL (Requirement Specification Language), um modelo abstrato semântico do sistema (ASSM) e um conjunto de ferramentas automatizadas.

- *UML – Unified Modeling Language*: linguagem para visualizar, especificar , construir e documentar os requisitos e informações de um sistema.
- *Viewpoints*: são um mecanismo que permite considerar aspectos do sistema percebidos por diferentes requerentes. Ao se analisar o sistema por vários aspectos obtém-se uma especificação mais adequada, a qual permite um melhor entendimento das necessidades dos usuários através de cada uma das atividades do processo.
- *Volere*: é um método completo de obtenção de requisitos, baseado nos casos de uso.

4.5 Qualidade em Software

De acordo com Wasselius e Ververs (1990), a qualidade em software é algo subjetivo. Ao consultar o dicionário, as definições encontradas que mais se afinam com o sentido buscado são:

“Superioridade, excelência de algo”
“Dote, virtude”

Segundo Wasselius a melhor definição pode ser dada por:

“Qualidade significa adequação a um fim específico.”

As características da qualidade compõem as peças básicas que a constroem.

Em relação à qualidade de software pode-se dizer que esta depende do grau no qual os atributos do software permitem especificá-lo em seu uso final. A qualidade pode ser considerada em dois aspectos:

- *Qualidade de adaptação* indica o grau de sucesso que o analista obteve no desenvolvimento de um produto a partir de determinados requisitos.
- *Qualidade do projeto* indica o grau de aceitação pelo cliente do produto desenvolvido.

Por esta razão a qualidade de software pode ser definida como a adaptação aos requisitos feita pelo analista. Formalmente a definição é dada pela norma ISO 8402-1986:

“Qualidade é a totalidade de aspectos e características de um produto de serviço que comportam sua habilidade em satisfazer necessidades declaradas ou implícitas.”

Esta definição retrata bem a subjetividade da qualidade, que não é julgada apenas pelas necessidades declaradas, mas pelas implícitas também, ou seja um produto que atenda todas as características declaradas pode ser considerado de baixa qualidade por não considerar necessidades subentendidas. Logo, a expectativa do usuário faz parte da definição de qualidade. Criar qualidade significa ‘agradar ao cliente’.

A tática de dividir-e-conquistar pode ser usada parcialmente quanto à qualidade. Para entender melhor como se constrói um produto de qualidade, a qualidade é decomposta em partes, porém nada assegura que a construção de partes de qualidade irá resultar num produto de qualidade.

A qualidade pode ser identificada por três componentes distintos:

- *componentes acessáveis objetivamente:* são os requisitos da especificação, devem englobar todas as características definidas para o produto. As preferências do cliente devem ser incluídas.
- *componentes acessáveis subjetivamente:* refere-se às preferências e expectativas individuais dos clientes.
- *componente não acessível:* o sistema deve se comportar de acordo em situações não previstas. O produto deve ser adaptável a futuras necessidades que virão, que ainda não são conhecidas.

4.5.1 Engenharia de Qualidade

Existem técnicas que propiciem o desenvolvimento de um software de qualidade? Sim, através da especificação de requisitos obtém-se a base de um desenvolvimento de software potencialmente bem sucedido, onde a não completude e a subjetividade são minimizadas (quality of design). A Engenharia de Qualidade oferece uma moldura para usar efetivamente conceitos e técnicas da Engenharia de Software.

A melhoria da Engenharia de Qualidade dá-se quanto maior for a participação dos clientes no processo de obtenção dos requisitos, pois os clientes e usuários são os juizes em última instância da qualidade. O uso de métodos que estimulem a participação destes afim de obter uma especificação que represente suas necessidades da forma mais completa possível é recomendado.

No entanto a Engenharia de Qualidade não se aplica somente à fase inicial do projeto, quando a Análise de requisitos é feita, segundo Pressman (1995) refere-se a todo o processo de desenvolvimento:

“A garantia de qualidade de software é uma atividade abrangente aplicada a cada passo do processo de Engenharia de Software, envolve procedimento para a efetiva aplicação de métodos e ferramentas, revisões técnicas formais, estratégias e técnicas de teste, procedimentos para o controle de mudanças, procedimentos para assegurar o cumprimento de padrões e mecanismos de medição e reportagem.”

Os fatores que afetam a qualidade de software focalizam três facetas distintas do mesmo:

Operação do produto:

- *Corretude:* o software faz o que o cliente quer?
- *Usabilidade:* ele foi projetado para o usuário?
- *Confiabilidade:* ele se comporta com precisão o tempo todo?
- *Eficiência:* ele rodará bem no hardware?
- *Integridade:* o acesso aos dados ou ao seu uso por pessoas não autorizadas é controlável?

Revisão do produto

- *Manutenibilidade:* esforço gasto para reparar erros num programa
- *Flexibilidade:* esforço gasto para alterar um programa em operação
- *Capacidade de teste:* esforço gasto para se testar um programa garantindo a execução das tarefas pretendidas

Transição do Produto

- *Portabilidade:* capacidade de migrar o programa para outra máquina

- Reusabilidade: capacidade de se aproveitar partes do código para ou de outros programas
- Interoperatividade : capacidade do programa em interagir com outro sistema

Neste trabalho, o foco se dirige a etapa inicial do processo de construção de software: elaboração da Especificação dos Requisitos, portanto os aspectos de qualidade de software pertinentes a esta etapa restringem-se a alguns dos fatores relativos a Operação do Produto. Estes fatores são: correção e usabilidade.

O ‘processo de desenvolvimento ideal’ ocorre quando a qualidade pode ser referenciada objetivamente através das relações existentes entre as características do produto e o conjunto de requisitos. É preciso aumentar a participação do componente objetivo da qualidade, tornando o processo de desenvolvimento ideal mais aplicável.

Etapas do processo de desenvolvimento ideal:

- Criação de um modelo do mundo real, este modelo permite que o analista compreenda o problema totalmente.
- Criação de um modelo do produto, neste modelo é estabelecido o conjunto de características necessárias (requisitos) que o produto deve conter, relacionando-as ao problema e seu contexto.
- Implementar o produto
- Avaliar o produto

4.6 Soluções e Abordagens

Na representação de um sistema, devem ser escolhidos vários modelos distintos que o representem. Um único modelo não é suficiente. Cada modelo escolhido deve ser feito em vários níveis de detalhamento, fornecendo visões múltiplas do sistema aos vários tipos de requerentes. A escolha do modelo tem uma profunda influencia em como o problema é abordado e na solução adotada. Os métodos de análise mais difundidos são a Análise Estruturada e a Análise Orientada a Objetos.

Na etapa inicial, as especificações ainda não devem ser completas, deixando margem aos ajustes vindos com a maturidade da compreensão do sistema. A construção de um protótipo permite ao requerente validar o sistema, pois a partir de um sistema parcial, mas concreto e visível (representado através do protótipo), torna-se mais fácil ao requerente perceber ou relacionar fatos relevantes que antes não haviam sido lembrados. Isso ocorre pois, além da dificuldade natural do requerente de compreender os diagramas e sentenças de requisitos afim de abstrair o futuro sistema, inicialmente o requerente tem focada sua atenção na resolução de um problema maior, básico ao seu negócio; à medida que este problema é tratado, problemas menores pendentes adquirem papel de destaque, tornando-se a nova preocupação do requerente. É importante destacar que os limites ou fronteiras do sistema em desenvolvimento são linhas tênues em constante expansão ou retração. Cabe ao analista orientar seu requerente sobre os custos desta expansão de fronteira, que constituem normalmente o fator limitante de maior expressão. O analista emprega os resultados finais da análise como base para o Documento de Requisitos de Software

Durante o processo de análise dos requisitos, podem ocorrer mudanças no ambiente do sistema, bem como nos objetivos a serem alcançados. Estas inevitáveis mudanças devem ser reconhecidas e antecipadas no Documento de Requisitos. Os requisitos não podem ser tratados estaticamente, pois sua natureza é dinâmica.

Veremos no próximo capítulo a condução do processo de análise e obtenção de requisitos segundo o Método Volere.

Capítulo 5

A Análise de Requisitos para o Projeto de um Sistema de apoio ao Ensino da Introdução à Computação

Neste capítulo as metodologias e técnicas estudadas são usadas visando a resolução de um problema específico: fornecer um modelo de implementação de software educativo que conduza efetivamente o processo de ensino/aprendizagem para a disciplina de Introdução à Computação.

Tendo como base e apoio os recursos pedagógicos e tecnológicos estudados, foi elaborada a análise de requisitos para a construção de um software educativo de apoio ao ensino da Introdução à Ciência da Computação.

Para a obtenção dos requisitos adotamos o método Volere – baseado em casos de uso, que fundamenta-se nos conceitos vistos no capítulo 4.

5.1 Análise de Requisitos pelo Método Volere

A Especificação dos Requisitos do Software Educacional podia ser feita sob qualquer uma das muitas técnicas vistas no capítulo 4. Ao selecionarmos uma das técnicas, procuramos por um método que fosse bem documentado, adaptável ao domínio da educação, pois não encontramos uma técnica específica a este domínio tão peculiar, e que tivesse uma seqüência bem definida e criteriosa. O Método Volere baseia-se nos casos de uso, tendo como finalidade conduzir ao entendimento de forma mais abrangente do domínio estudado. Dos métodos vistos foi o que melhor correspondeu às necessidades.

O Volere, brevemente citado no capítulo anterior, será melhor explicado conforme ocorrer o desenvolvimento da técnica. Um esquema simplificado do método é mostrado na Figura 1.

Um caso de uso descreve uma parte do serviço, do ponto de vista do usuário. Os requisitos funcionais descrevem as ações pertinentes ao sistema. Os requisitos não funcionais descrevem a experiência obtida pelo usuário durante a utilização do sistema, ou seja os requisitos não funcionais são as características do sistema.

Iniciemos pois a aplicação da técnica, levando em consideração a pesquisa feita nos capítulos iniciais desta dissertação.

Figura 1 - Esquema do Método Volere

5.1.1 Levantamento inicial das características do produto a ser desenvolvido

Antes de desenvolver um produto é necessário verificar sua viabilidade, identificando suas principais características.

Objetivo do Software

Desenvolver um ambiente computacional que favoreça a aprendizagem da programação em linguagem imperativa. O projeto é um software educativo de ensino da programação. Atua

como ferramenta auxiliar ao professor nas aulas de Introdução à Computação. Este software não tem por objetivo substituir o professor, mas sim ser um recurso que auxilie os alunos a compreender e a estudar a matéria fora do horário de aula. Seu uso é independente do uso do compilador, mas pode ser usado como referência da linguagem de programação e ajudar nas dúvidas decorrentes da prática da programação.

Stakeholders (*peças que conhecem o domínio estudado, ou que possuem algum tipo de interesse no produto*)

Alunos que estão cursando a matéria de Introdução a Computação, alunos que já fizeram a matéria, professores de Introdução a Computação, projetistas de softwares educativos ligados ao ensino da programação e psicólogos educacionais.

Cliente (*aquele que financia o desenvolvimento do produto*)

USP-IME, representado pelo orientador da dissertação.

Freguês (*aquele que estando o produto pronto, o compra*)

Professores do Departamento de Ciência da Computação do IME e alunos de Introdução a Computação.

Usuário (*aquele que usa o produto*)

Alunos de Introdução a Computação e professores do Departamento de Ciência da Computação do IME.

Restrições

Apesar de termos visto vários ITS nos capítulos anteriores, vimos também as dificuldades de sua implementação. Este projeto não pretende desenvolver um ITS ou usar Inteligência Artificial, mas idéias e benfeitorias advindas dos ITS, possíveis de implementação, serão adotadas, bem como o uso de recursos tecnológicos: multimídia e hipertexto, e recursos pedagógicos, vistos no capítulo 2.

Glossário de termos técnicos

Como o domínio é conhecido pelo analista, pelo cliente e pelo freguês não se faz necessário definir os termos usados.

Custo

O tempo gasto pela aluna de mestrado.

Riscos

Os riscos envolvidos são muitos, devido à interdisciplinariedade do projeto e à ausência de modelos significativos, este produto não tem similares. O maior desafio deste projeto é apresentar uma proposta que seja viável e de qualidade.

5.1.2 Intenções do produto

Propósito *(o que o produto deve fazer)*

Auxiliar o entendimento de estruturas básicas da programação, num ambiente interativo e cooperativo, baseado no construtivismo.

Vantagem *(quais vantagens virão com o uso do produto)*

Facilitar o entendimento da programação, diminuindo o número de repetências.

Medida *(medida das vantagens, meta)*

As notas devem aumentar em relação às notas sem o uso do software.

Viabilidade *(o esforço de desenvolver o produto é maior que as vantagens ?)*

Devido ao avanço das ferramentas de desenvolvimento de software multimídia, o esforço de desenvolvimento será minimizado.

Execução *(o produto será capaz de atingir sua meta?)*

O presente projeto restringe-se à Análise dos Requisitos de Software, não faz parte deste projeto implementar o sistema, portanto a eficácia do produto final: o software educacional, não poderá ser medida.

Realizável *(os desenvolvedores são habilitados para desenvolver o produto?)*

Os capítulos anteriores desta dissertação, ao identificarem o problema e os recursos necessários a sua resolução, proporcionam uma base estável para, seguindo os passos do método Volere, tornar possível a realização do produto.

5.1.3 Stakeholders

Como vimos são as pessoas que estão direta ou indiretamente relacionadas ao sistema.

a) Identificação

Alunos que estão cursando a matéria de Introdução a Computação

Possuem em geral experiência como usuário de computador, sabem usar o computador mas muitas vezes sem a compreensão formal de sua estrutura geral. Parte dos alunos ao entrar na faculdade, já conhece alguma linguagem de programação, (em geral Visual Basic, Access ou Delphi) mas o restante dos alunos terá seu primeiro contato ao cursar esta disciplina.

Em geral, sabem navegar pela Internet e trocar e-mail. Estão motivados a aprender a programar, que é uma das atividades principais da profissão.

Alunos que já fizeram a matéria

Podem colaborar com sua experiência, pois são conscientes das dificuldades iniciais na aprendizagem da programação.

Professores de Introdução à Computação

Possuem estilos pessoais bem distintos de ensinar. Devido a minha participação no Projeto Mac-Multimídia, tive oportunidade de inteirar-me nas idéias e opiniões dos professores. A maioria acredita que seja possível criar um software educacional para ensinar programação, só não sabem como deve ser este software.

Orientador

Espera que o desenvolvimento seja feito segundo a Engenharia de Requisitos, que a partir de um estudo de várias possíveis soluções, seja obtida uma solução plenamente justificada que se adeque ao problema. Quer um critério que meça a eficácia da solução proposta.

Projetistas de softwares educativos ligados ao ensino da programação

Por já terem percorrido o caminho, registraram as iniciativas que deram resultados positivos bem com as que não corresponderam as intenções iniciais. Possuem grande conhecimento das dúvidas mais freqüentes e sua experiência pode ser valiosa ao projeto.

Psicólogos Educacionais

Conhecem o lado humano do aprendiz: como ocorre o aprendizado, quais as técnicas que dão resultado, o que não se deve oferecer ao aluno, como motivar o aluno ao aprendizado e como não desmotivar o aluno.

b) O que esperam do Produto

Alunos que estão cursando a matéria de Introdução a Computação

Os alunos com dificuldade possivelmente irão se interessar pelo produto, os que já possuem conhecimento prévio preferirão usar o compilador diretamente. O visual da interface deve ser atrativo e o programa deve ajudar a descobrir erros em trechos de código. Os alunos esperam um programa que seja interativo e afinado às suas necessidades de aprender a programar. Ao ocorrer uma dúvida ou dificuldade com a programação, o programa deve poder conduzir o aluno a uma solução. Esperam que o uso do software esteja incorporado ao esquema didático do professor, e que o professor tire suas dúvidas também fora da sala de aula. Esperam que o software seja um prolongamento da aula - um assistente do professor, e que tenha comunicação com este. Esperam também poder ver o que os colegas fizeram e comunicarem-se eletronicamente.

Alunos que já fizeram a matéria

Esperam que o produto seja capaz de resolver as dúvidas maiores por que passaram. Irão testar o software e darão sugestões de como o software ficaria melhor segundo sua preferência pessoal.

Professores de Introdução a Computação

Não sabem bem o que o software deve ter ou como integrá-lo à sua aula. No início deixarão que os alunos experimentem o software e caso a aceitação seja boa, irão aos poucos se acostumando e eventualmente irão interagir propondo melhorias ao software.

Orientador

Espera que venha efetivamente a contribuir com o ensino de MAC-110, auxiliando o professor na retaguarda através do reforço de conceitos e como exercitador da matéria. Precisa de critérios que meçam a eficácia do projeto assegurando que o resultado obtido é positivo.

Projetistas de softwares educativos ligados ao ensino da programação

Que o software contenha as idéias boas de seu projeto, mas que seja capaz de corrigir os erros ou lacunas detectados.

Psicólogos Educacionais

Que o software não deve dirigir o aprendizado, sendo que o aluno deve ter as rédeas do processo. Quanto menor for a influência do software nas decisões do aluno, mais adequado este será. O software deve dar mecanismos que permitam ao aluno a auto-avaliação de seu trabalho, levando o próprio aluno a refletir sobre seu processo de busca do conhecimento, aprendendo a corrigir-se quando necessário na procura da solução.

5.1.4 Sistemas Adjacentes

São os sistemas que possuem conexão direta com o sistema em desenvolvimento (o projeto). Os sistemas adjacentes podem ser ativos, autônomos ou cooperativos. Parte dos stakeholders são sistemas adjacentes.

Ativo (*age dinamicamente, podendo interagir ou participar do serviço, normalmente são pessoas*)

- Alunos que estão cursando a matéria de Introdução a Computação
- Professores de Introdução a Computação

Autônomo (*é um sistema externo que interage indiretamente com o serviço. Ele age independentemente, mas se relaciona num fluxo unidirecional, sem esperar uma resposta imediata ou uma ação*)

- não há

Cooperativo (*é um sistema que possui um comportamento previsível quando solicitado, normalmente um equipamento ou um programa de computador.*)

- Internet

As interações entre os sistemas adjacentes e o sistema são identificados como ‘eventos’. Ao ocorrer um evento o sistema deve agir segundo uma pré-programação, gerando uma resposta pré-planejada a este evento. Eventos ocorrem também devido à passagem do tempo.

5.1.5 Pequenos Pedacos do Software

São as primeiras idéias de serviços que o sistema deve conter. Trata-se apenas de uma lista de tarefas, antes de procurar alguma forma de classificação.

Figura 2 - Pequenos pedaços do software

5.1.6 Diagramas do Modelo Atual e Modelo Proposto

Como vimos no capítulo 4, a representação do sistema em um modelo é fundamental para entendimento do mesmo, servindo como base à construção do novo sistema

A partir da identificação dos elementos pertencentes ao sistema e de toda a pesquisa feita nos capítulos anteriores, são feitos os diagramas:

- *do sistema atual*, que mostra como é efetivamente feito o trabalho e as relações entre os vários elementos, mesmo que o trabalho seja sem o uso do computador (Figura 3);
- *do sistema proposto*, onde são mostradas as novas formas de se conduzir o serviço, a partir do uso do sistema proposto (Figura 4).

Figura 3 - Modelo do Sistema Atual

A proposta deste software educativo não é substituir o professor, como ocorre nos ITS, mas de ser uma ferramenta auxiliar ao curso, fora do horário de aula. A figura do professor tem um papel relevante no novo sistema, é ele quem define o ritmo do curso e faz a ponte entre as aulas presenciais.

Figura 4 - Modelo do Sistema Proposto

No entanto, o software poderá ser usado, de modo parcial, sem a participação do professor. Nesta configuração toda a parte de colaboração entre alunos fica prejudicada, sendo usado principalmente o ambiente semi-dirigido de exercícios.

5.1.7 Diagrama de Contexto

O diagrama de contexto mostra as conexões entre o sistema e o mundo exterior. Estas conexões são obtidas através das trocas de mensagens com os sistemas adjacentes (Figura 5).

Cada conexão é gerada por um evento. Estudando as conexões, pode-se estudar os eventos. As conexões são as trocas de serviços entre os sistemas adjacentes e o sistema proposto.

Um evento ativa um ou mais pedaços do software, gerando uma resposta do sistema. Esta resposta, dada pelo sistema a um evento, será estudada nos próximos itens.

Apesar de não estar incluído no método Volere, acrescentei um diagrama de contexto extra (Figura 6), com os sistemas adjacentes presentes apenas durante o desenvolvimento do sistema. Identifiquei dois tipos de sistemas adjacentes nesta categoria:

- sistemas contendo a base de conhecimento sobre o domínio, como livros ou pesquisas;
- sistemas que moldam a interação com o usuário, como os recursos tecnológicos e pedagógicos estudados.

Estes sistemas adjacentes não geram eventos, mas são tão fundamentais na construção de sistemas interativos como os primeiros.

Figura 5- Diagrama de Contexto

Figura 6- Diagrama de Contexto Extra

5.1.8 Lista de Eventos de cada Sistema Adjacente

Como vimos os sistemas adjacentes estabelecem conexões com o sistema, através da geração de eventos. Os eventos iniciados por cada um dos sistemas adjacentes estão listados abaixo:

Alunos que estão cursando a matéria de Introdução a Computação

- praticam exercícios de programação;
- comunicam-se com o professor e os demais colegas;
- obtêm referência aos comandos da linguagem C;
- obtêm suas estatísticas de aprendizado;
- obtêm referências bibliográficas;
- identificam-se no início

- obtêm listagem dos exercícios propostos e resolvidos.

Professor de Introdução à Computação

- verifica a lista de alunos que usam o software;
- acompanha estatísticas da avaliação do aprendizado do aluno e da turma;
- edita exercícios;
- troca comunicações com os alunos;
- atualiza referências bibliográficas e ensina os alunos a buscar novas fontes de informação;
- disponibiliza as notas de aula e exercícios propostos;
- identifica-se no início;

Internet

- permite troca de mensagens entre alunos e professor

5.1.9 Lista de Eventos

Nesta lista os eventos são reunidos, pois em geral há sobreposição de eventos, ou seja dois sistemas adjacentes podem gerar o mesmo tipo de evento. Para cada evento é identificado o tipo de informação trocada com o sistema e o sentido do fluxo (e: entrada, s: saída)

Nome do Evento	Entrada e Saída
1) Identificação do usuário	Dados do usuário (e)
2) Acompanhamento das estatísticas de aprendizado do aluno e da turma (professor)	Relatório de aproveitamento do aluno e da turma(s)
3) Edição de exercícios (professor)	Novo exercício (e)
4) Comunicação entre professores e alunos	Dúvidas (e) e Respostas (s)
5) Atualização das referências bibliográficas	Referências Bibliográficas (e)
6) Obtenção das notas de aula e dos exercícios propostos	Listagem das notas de aula e exercícios propostos(s)
7) Atividades praticas com exercícios	Diálogo com o tutor (e/s)
8) Obtenção de referências da linguagem C	Referência da Linguagem C (s)
9) Obtenção de referências bibliográficas	Relatório de Referências Bibliográficas (s)
10) Obtenção de estatística de seu desempenho (aluno)	Relatório de Aproveitamento (s)
11) Obtenção de listagem dos exercícios resolvidos	Lista de Exercícios Resolvidos (e/s)

5.1.10 Resposta do Sistema aos Eventos - Cenários

Neste item são discutidos os *cenários* ou *casos de uso* de cada evento. Os casos de uso são as ações feitas pelo sistema em resposta a um evento, gerado por um *ator* – ou sistema adjacente (usuário), sob o ponto de vista externo ao sistema, ou seja não serão analisados procedimentos de implementação de software, mas sim como o sistema interagirá com o sistema adjacente que disparou o evento.

A abordagem por *casos de uso* na análise de requisitos, conduz à obtenção dos requisitos para cada um dos casos de uso. É uma forma de se dividir o sistema em partes menores, de modo eficiente, pois os casos de uso representam cada um dos serviços a ser feito pelo sistema em estudo. Os requisitos obtidos desta forma referem-se ao ator, ou seja o que o sistema deve fazer para permitir que haja uma interação satisfatória com o ator.

A seguir são descritos os cenários para cada um dos eventos contidos na lista de eventos:

1) *Identificação do usuário*

O sistema se comporta de modo distinto de acordo com o tipo de usuário. O professor possui acesso completo a todas as áreas do projeto, enquanto que os alunos têm acesso parcial, restrito às comunicações da sua sala

A identificação verifica se é professor ou aluno já inscrito. Se for inscrito carrega a ficha correspondente e inicia o rastreamento das principais ações do aluno. Se não for inscrito oferece uma ficha de inscrição ao aluno.

2) *Acompanhamento das estatísticas de aprendizado do aluno*

Ao identificar o aluno, o sistema inicia um rastreamento de suas principais ações, como tempo de utilização do sistema, exercícios resolvidos, etc., registrando-as. A partir destes dados, são levantados dados estatísticos como média e desvio padrão, da classe e individualmente por aluno. Estes dados têm por finalidade dar um retorno ao professor do grau de entendimento, de como os alunos estão acompanhando o curso, dos pontos onde há necessidade de uma revisão.

3) *Edição de exercícios*

Haverá uma área de edição, onde o professor com base no acompanhamento das estatísticas de aprendizado, possa inserir novos exercícios, mais adequados às dificuldades de

uma turma em particular. O professor pode também utilizar-se de um banco de dados de exercícios contidos no sistema, para indicar aos alunos quais são os exercícios recomendados.

4) Comunicação entre professores e alunos

Como existem vários professores e várias turmas, a princípio cada turma permanece isolada das demais, ou seja o aluno que estiver na turma A com o professor X, no projeto ele pertencerá a esta mesma turma, com este mesmo professor. A idéia é que o projeto sirva como um arquivo de todas as mensagens trocadas entre a turma, devido ao teor das mensagens estar intimamente ligado ao contexto da aula dada por aquele professor.

No caso de alunos avulsos, estes pertencerão a uma classe virtual e trocarão mensagens com um professor voluntário.

Existem alguns tipos de comunicação possíveis:

- Professor - alunos
- Professor - um aluno
- Aluno - professor
- Aluno - colegas

5) Atualização das referências bibliográficas

O professor, sempre que necessário, deve atualizar as referências bibliográficas do curso, bem como ensinar os alunos a se atualizarem. Nesta área haverá também referências a artigos, resumos de artigos e links para sites sobre programação. Os alunos podem incluir referências também, em área específica. É importante para o aluno adquirir o hábito de buscar suas próprias informações, nesta área do sistema, além do material previamente selecionado pelo professor como referências a livros e revistas, o aluno será estimulado a pesquisar novas fontes, e anexá-las, podendo inclusive escrever sua opinião sobre o novo material, disponibilizando-o aos demais colegas. Haverá uma 'moldura' a fim de padronizar a apresentação desta informação, pois as atualizações serão feitas por várias pessoas.

6) Obtenção das notas de aula e de exercícios propostos

A página do professor têm por finalidade situar o aluno no curso que está sendo dado. É um registro do andamento do curso. Deve conter o planejamento das aulas do semestre, com data e matéria a ser ministrada e possíveis resumos dos conteúdos de cada aula. Esta área pode

ser usada também para divulgação das notas das provas. Os exercícios propostos de cada aula, podem ser enviados por e-mail aos alunos, ou relacionados na página do professor, que manterá o registro por aula, permitindo que exercícios propostos para aulas anteriores sejam consultados.

7) *Atividades práticas com exercícios*

Esta atividade é o ponto central do novo sistema. Aqui serão usados os recursos pedagógicos, numa abordagem construtivista de interação com o aprendiz. É proposto um ambiente de trabalho semi-dirigido (item 5.1.14) no qual serão apresentados exercícios elaborados segundo técnica desenvolvida nesta dissertação e apresentada no item 5.1.12.

8) *Obtenção de referências da linguagem C*

Haverá um guia de referência da linguagem C, com uma breve descrição do tópico, seguido de exemplos. Este guia será relacionado ao ambiente de trabalho semi-dirigido, e no caso de não acertos do aluno na realização de um exercício, será sugerida a leitura do tópico relacionado deste guia.

9) *Obtenção de referências bibliográficas*

O aluno pode obter um relatório das referências bibliográficas preparadas pelo professor ou por colegas.

10) *Obtenção de estatística de seu desempenho (aluno)*

O aluno poderá acompanhar seu desempenho comparando seus resultados com a média da classe. Note que não será possível um aluno verificar o desempenho de outro, por razões de privacidade, através da identificação inicial o sistema é capaz de restringir o acesso.

11) *Obtenção de listagem dos exercícios resolvidos*

Os exercícios resolvidos podem ser postos nesta área pelos alunos, será interessante como forma de comparação dos diversos modos possíveis que existem para resolver um problema. O professor pode até fazer uma comparação entre as soluções, destacando pontos positivos e negativos das soluções obtidas.

5.1.11 Observações do Serviço

Vamos neste ponto fazer uma breve recapitulação do processo de obtenção de requisitos:

- 1) O Diagrama de Contexto mostrou os sistemas adjacentes e sua interação com o sistema proposto através dos eventos;
- 2) Os eventos foram identificados e listados;
- 3) Para cada evento foi descrito um cenário, ou seja como o sistema reage em resposta ao evento.

Do todo - diagrama de contexto, obteve-se a divisão em partes – cenários. Estas partes foram separadamente estudadas. E num movimento contrário – das partes para o todo, o sistema será reconstruído.

Segundo os autores do Método Volere :

“Os stakeholders são a fonte dos requisitos, os analistas são os tradutores das sentenças em especificações do produto” (Robertson e Robertson, 1999).

E os tipos de requisitos obtidos através dos usuários são:

- **consciente:** são as melhorias que o usuário acha que o trabalho deve ter;
- **inconsciente:** o usuário não menciona devido ao fato dele conhecer tão bem o requisito que supõe que todos também o conheçam;
- **não-sonhadores:** são as coisas que os usuários irão solicitar quando descobrirem que é possível ser feito.

Segundo a técnica Volere, o melhor modo de se entender como o trabalho é feito é observar o usuário trabalhando, e fazer algumas perguntas. A entrevista é uma técnica de se obter requisitos, o entrevistado é informado sobre o contexto do trabalho, são preparadas perguntas e diagramas sobre o trabalho.

A partir disto foram feitas observações e questionários para os stakeholders mais envolvidos com a utilização do software :

- formulário de observação para os alunos que cursam a matéria (Anexo I)

A seguir é feita uma breve descrição do objetivo e dos resultados obtidos:

Observação dos alunos que cursam a matéria

Objetivo: observar o ambiente de trabalho atual dos alunos, fora e dentro do laboratório.

Contexto do trabalho: integrar a aula teórica às atividades práticas de programação.

Metodologia: os alunos são observados em duas situações: fora e dentro do laboratório. Numa sala é dado um problema aos alunos, e observada a resolução. Os alunos são divididos em grupos de dois; através do diálogo as etapas de desenvolvimento são melhor observadas pelo pesquisador. Após a resolução ser feita em papel e lápis os alunos lêem as observações feitas, comentando-as, e completando-as. A 2ª situação ocorre no laboratório, onde os alunos vão usar o compilador, testando o problema resolvido anteriormente. No final, os alunos respondem um questionário sobre sua experiência anterior no uso de computadores.

Observações: os alunos de MAC-110 foram convidados a participar como voluntários. Reuni um grupo de 5 alunos, que receberam o enunciado do problema e os observei enquanto desenvolviam sua solução. Os alunos resolveram com rapidez, alegando que haviam acabado de resolver um Exercício-programa, o que lhes dera presteza com a programação.

Resultados Obtidos: apesar do baixo número de alunos, observei algumas falhas no processo:

- o problema proposto era inadequado ao estágio corrente do curso, era fácil demais, não propiciando um nível de desafio compatível à observação de dificuldades. Para um melhor resultado, eu deveria ter entrado em contato com o professor da turma e saber exatamente o que estava sendo dado no momento;
- os alunos que aceitaram ser voluntários, tinham experiência anterior em programar, por esta razão sentiram-se mais confiantes em se expor a esta observação. Os alunos com maior dificuldade não aceitaram participar, esquivando-se;
- alguns alunos que aceitaram participar faltaram no dia da apresentação.

A intenção de fazer observações e questionários, proposto pelo Volere, tem por objetivo conhecer melhor a operacionalização de uma tarefa ou serviço, mas no contexto educacional o processo é individual e altamente mutável, a tarefa é alterada a cada semana.

Devido às dificuldades encontradas, resolvi não prosseguir com as observações. Procurando obter estas informações por outras fontes:

- minha experiência como docente e aluna de programação;
- Curso de C da UFMG

Conclusões sobre os questionários

O resultado obtido ficou muito aquém do esperado, pois não consegui obter as principais dúvidas dos alunos.

Analisando os resultados, cheguei a conclusão que a técnica de fazer questionários é pouco adequada para a obtenção de requisitos específicos de softwares educativos, pois é custoso em tempo do analista e boa vontade dos voluntários, sem gerar resultados satisfatórios.

5.1.12 Como usar os recursos pedagógicos na obtenção de exercícios adequados

Das teorias de aprendizagem, vistas no capítulo 2: o behaviorismo e o construtivismo, acreditamos que o construtivismo seja mais adequada ao ensino da programação, pois favorece o entendimento de propriedades ou características de novos elementos, suas relações com os elementos já conhecidos e inter-relações, ao passo que o behaviorismo busca um comportamento ou resposta adequada a um estímulo específico, mas sem procurar entender como o processo de cognição ocorre.

Como vimos no capítulo 2, os agrupamentos referem-se a operações realizadas com classes lógicas. Tratam do relacionamento existente entre estas.

Para poder aplicá-lo ao nosso domínio da programação, é preciso identificar as classes que o compõem. A divisão do conteúdo a ser aprendido em classes propicia :

- *entendimento de uma classe*, discernimento de pertinência de um elemento a uma determinada classe. Uma vez classificado o elemento, as características gerais da classe aplicam-se a ele;
- *relação existente entre as classes*, como interação elementos de classes distintas, quais são as operações possíveis;
- *transformações sobre relacionamentos*, como podem ser alteradas as características de relacionamento entre elementos, levando a novas relações, novas possibilidades de se obter soluções.

Sendo assim, o sistema aqui proposto tem um enfoque distinto dos softwares educacionais voltados ao ensino da computação vistos no capítulo 3. Tais sistemas partem, desde a primeira

aula, da programação de uma função, com todos os elementos que a compõe. Tal aproximação dificulta a classificação pelo aluno dos vários elementos envolvidos.

O que está sendo proposto é dividir por partes os elementos de um programa, trabalhando-os da forma mais concreta possível, pois é através da ação do sujeito sobre um elemento (acomodação, cap. 2), que torna este elemento externo e desconhecido um elemento interno e conhecido (assimilação).

Figura 7 - Aquisição de conhecimento

Assim, antes de resolver um exercício, o aluno deve conhecer um pouco melhor o que faz cada elemento do programa, como se declaram variáveis, como se declara o

protótipo de cada função. Para capacitar o aluno em tais habilidades está sendo proposto um ambiente de trabalho semi-dirigido, onde é pedido que seja escrito em linguagem computacional, não um problema, mas uma parte deste: uma sentença em linguagem natural.

5.1.12.1 Classes Propostas

Os elementos básicos das linguagens de programação imperativas, foram classificados em classes e subclasses que servirão de base na sequência de ensino da matéria. O critério para classificação baseou-se nos índices de livros de ensino de programação (Kernighan e Ritchie, 1990; Wirth, 1985; Ferrer et al., 1999; Curso C-UFMG, cap. 3) e em análise feita por mim.

A observância desta sequência é fundamental. Existe uma ordem de apresentação das classes :

- a) Variáveis
- b) Operadores

- c) Expressões
- d) Comandos
- e) Funções
- f) Ponteiros

Esta ordem evita que haja sobreposição de conceitos e limita o aprendizado a um conceito novo por vez. Assim ao serem introduzidas as expressões, o aluno deve já ter assimilado o que são variáveis e operadores.

Os esquemas mostram as classes e suas sub-classes, assim são apresentadas as propriedades gerais da classe e depois as especializações de suas sub-classes. As restrições, ou pré-requisitos à uma classe são relacionados, indicando a ordem a seguir.

a) Variáveis

b) Operadores

Restrição:

Classe: operadores → Pré-requisito: classe variáveis

c) Expressões

Restrições:

Classe : expressões aritméticas → Pré-requisitos: classe variáveis e classe operadores aritméticos.

Classe : expressões condicionais → Pré-requisitos: classe variáveis, classe operadores lógicos e classe operadores relacionais.

d) Comandos

Restrições:

Classe : atribuição → Pré-requisito: classe expressões aritméticas.

Classes : repetição e seleção → Pré-requisitos: classe expressões aritméticas e classe expressões condicionais.

e) Funções

Restrições:

Classe : funções criadas pelo aluno → Pré-requisitos: classe comandos e classe variáveis.

f) Ponteiros

Restrições:

Classe : ponteiros → Pré-requisito: classe variáveis.

5.1.12.2 Elaboração de Exercícios

Para elaborar os exercícios, serão seguidas as Leis do Agrupamento, os Agrupamentos e as Operações Interproposicionais vistas no capítulo 2. Estas leis e lemas, segundo a teoria construtivista propiciam ao aluno passar pela acomodação e atingir a assimilação do novo elemento estudado, compreendendo o próprio processo de aprendizagem, isto é, aprendendo a pensar e a agir de forma lógica e refletida, evitando repetições desnecessárias, comportamentos impulsivos ou seleção por ensaio e erro.

O cruzamento das leis da teoria piagetiana com as classes identificadas no domínio da programação, não será feito de forma exaustiva, serão destacados os cruzamentos mais relevantes a fim de mostrar como o processo pode ser conduzido.

É apresentado a seguir uma proposta de elaboração de exercícios. Para cada classe será feita a análise dos seguintes tópicos:

- *1º Passo: as dificuldades mais freqüentes encontradas pelos alunos*, esta etapa busca identificar os erros mais comuns para criação de exercícios específicos sobre o conceito. A obtenção dos erros e dúvidas mais comuns dos alunos, foi extraída da lista de discussão do Curso de C – UFMG (capítulo 3) e relatada no Anexo II;

- *2º Passo: as características gerais da classe*, serão usadas tanto para preparar os exercícios como para compor o Guia de referência da linguagem;
- *3º Passo: as particularidades de cada sub-classe*, cada sub-classe terá seu tratamento individualizado, destacando somente as características que a especificam em relação à sua super-classe;
- *4º Passo: os agrupamentos que promovem um melhor entendimento da classe*, o cruzamento das características da classe com os agrupamentos não será feita de modo exaustivo. Serão considerados somente os agrupamentos que melhor promoverem o entendimento das características de uma classe;
- *5º Passo: os exercícios para a classe*, serão propostos exercícios para cada cruzamento classe x agrupamento;
- *6º Passo: os exercícios para a sub-classe*, cada sub-classe terá o mesmo tratamento da classe, tendo seus exercícios específicos.

Vejamos então alguns casos de como se dá o processo:

Exercícios sobre a Classe Variáveis

1º Passo: Dificuldades encontradas pelos novatos:

- 1) identificação correta do tipo;
- 2) como é a alocação de memória no union;
- 3) uso de uma variável antes de declará-la;
- 4) atribuição a uma variável de um tipo distinto ao da variável;
- 5) ler o valor de uma variável não inicializada;
- 6) utilização de números inteiros acima do limite;
- 7) expressões aritméticas com tipos distintos;
- 8) uso de acentos ao nomear uma variável;
- 9) variável extern;
- 10) variável de ambiente (global);
- 11) uso dos modificadores de tipo: long, short;
- 12) uso da vírgula como separador decimal;

13) acesso a elementos de uma string;

2º Passo: Características gerais da classe:

- A declaração de uma variável é definitiva, a variável não pode ter o tipo de dado alterado.
- Todas as variáveis devem ser declaradas antes de usadas.
- A declaração de uma variável implica em reservar uma área da memória.
- A escolha de um tipo de dado deve basear-se em:
 - capacidade de armazenar valores característicos da informação,
 - uso adequado da memória disponível, não superestimando os limites da informação.
- O nome da variável é definido segundo alguns critérios e restrições:
 - o nome pode conter letras, dígitos e _ (subscrito);
 - o 1º caracter do nome deve ser uma letra;
 - um nome pode ter no máximo 31 caracteres;
 - as letras minúsculas e maiúsculas são distintas;
 - não podem ser usadas palavras reservadas na declaração de variáveis;
 - não podem ser usados acentos ou ç;
 - procure dar um nome que seja significativo a finalidade da variável;
 - a prática tradicional do C é usar minúsculas para nomes de variáveis e maiúsculas para constantes simbólicas.

3º Passo: Características da Sub-classe: Tipos primitivos

- O C possui 5 tipos primitivos de dados: int, float, double, char e void.
- Os tipos de dados numéricos são: int, float e double.
- O tipo char armazena caracteres: números, letras ou símbolos.
- O tipo void indica a ausência de um tipo de dado.
- O modificador short aplica-se ao tipo int alterando suas características.
- O modificador long aplica-se aos tipos int e double alterando suas características.

- Os qualificadores `unsigned` e `signed` aplicam-se aos tipos `int` e `char` alterando suas características.
- Os tipos `unsigned` são sempre números positivos ou zero.
- Uma variável do tipo `int` só aceita dados do tipo `int`.
- Uma variável do tipo `float` aceita dados dos tipos `int` e `float`.
- Uma variável do tipo `double` aceita dados dos tipos `int`, `float` e `double`.

4º e 5º Passos: Agrupamentos x Exercícios da Classe Variáveis

Leis do Agrupamento	Classe Variáveis - Conceitos	Dificuldades dos novatos	Exercício Proposto	Possíveis Soluções
Lei da composição (LA1) ¹ expressa a possibilidade de composição de dois elementos em um terceiro que os contenha.	O resultado da operação de duas variáveis de um tipo pode armazenar numa terceira variável.	Atribuição a uma variável de um tipo distinto ao da variável.	Declare a variável soma, que guardará o resultado de operações entre nota1 e nota2: float nota1; float nota2;	incluir a declaração: float soma;
Lei da reversibilidade (LA2) expressa a possibilidade de que toda operação admite uma inversão.	A declaração de uma variável é definitiva, a variável permanece no mesmo tipo de dado durante todo o programa.	Redeclaração de uma variável.	Altere o código para que fique correto: int classe; int tipo= 3; char classe='a';	excluir a declaração: int classe;
Agrupamento VIII (Ag VIII): da multiplicação co-unívoca de classes.	Variáveis só podem ser lidas depois de declaradas e inicializadas.	Uso de uma variável antes de declará-la.	Corrija o código abaixo: int ano_inicial; int ano_final; ano_final= ano_inicial + 20;	modificar: int ano_inicial = 1980; ou acrescentar: ano_inicial= 1980
Não há.	O separador decimal é o ponto.	Uso da vírgula como separador decimal.	Corrija o código abaixo: float altura, peso; int idade; altura= 1,75; peso= 67,3; idade= 19;	altura= 1.75; peso= 67.3;
Não há.	Não podem ser usados acentos ou ç.	Uso de acentos ou ç.	Corrija o código abaixo: int n_irmaos= 5; int n_filhos= 3; int n_netos= 15;	int n_irmaos= 5;

¹ As referências referem-se às Lei do Agrupamento, Agrupamento e Operações Interproposicionais descritas no capítulo 2

4º e 6º Passos: Agrupamentos x Exercícios sub-classe: Tipos Primitivos

Leis do Agrupamento	Classe Variáveis - Conceitos	Dificuldades dos novatos	Exercício Proposto	Possíveis Soluções
Lei da associatividade (LA3) expressa a possibilidade de se conseguir o mesmo resultado através de diversas operações alternativas.	Os tipos numéricos permitem uma maior flexibilidade na declaração: short int, int	Identificação correta do tipo; Uso dos modificadores de tipo: short e long.	Declare duas variáveis, cada uma com uma tipo de dado, para as seguintes informações: idade (idade1, idade2) peso (peso1, peso2) temperatura (temp1, temp2)	idade :int, short int, unsigned int peso: float, double temperatura: int, short int, signed int, float
Agrupamento II (Ag II): soma secundária de classes.	O armazenamento de números inteiros superiores ao limite de long int é feito nos tipos float ou double.	Identificação correta do tipo; Utilização de números inteiros acima do limite.	Declare uma variável que armazene nº de telefone (8 dígitos).	long int, long, unsigned long, unsigned long int; double; float ;
Agrupamento V (Ag V): soma de relações assimétricas.	Há uma sequência crescente de capacidade de armazenamento nos tipos de dados numéricos: short int < int < long int < float < double	Identificação correta do tipo; Uso dos modificadores de tipo: short e long.	O valor a ser armazenado não é comportado pelo tipo short int, declare uma variável capaz de armazená-lo.	int valor;

Exercícios sobre Expressões

1º Passo: Dificuldades encontradas pelos novatos:

- 1) como usar o operador aritmético mod;
- 2) como usar o operador lógico de negação;
- 3) como relacionar vários operadores numa expressão;
- 4) uso da atribuição no lugar do operador de igualdade;
- 5) uso de operadores unários e binários.

2º Passo: Características gerais da classe:

- Expressões são compostas por variáveis, constantes e operadores.
- A resolução de uma expressão obedece a ordem de avaliação definida pelo grau de precedência dos operadores nela contidos.
- Existe uma relação de precedência para todos os operadores.
- O uso de parênteses altera a precedência dos operadores.
- A resolução de uma expressão que contenha tipos de dado distintos nem sempre é possível, somente algumas conversões são possíveis como:
 - os char e short int são convertidos para int; os float são convertidos para double,
 - para pares de operandos de tipos diferentes: se um deles é long double o outro é convertido para long double; se um deles é double o outro é convertido para double; se um é long o outro é convertido para long; se um é unsigned o outro é convertido para unsigned.
- o uso do modelador (cast) força a conversão do tipo de dado resultante de uma expressão em outro tipo de dado.

3º Passo: Características da Sub-classe: Expressões Condicionais

- Uma expressão condicional tem como resultado dois estados: verdadeiro (1) ou falso (0).
- Uma expressão condicional é composta por variáveis, constantes e operadores relacionais e/ou lógicos.

4º e 5º Passos: Agrupamentos x Exercícios Classe Expressões

Leis do Agrupamento	Classe Expressões – Conceitos	Dificuldades dos novatos	Exercício Proposto	Possíveis Soluções
Agrupamento V (Ag V): soma de relações assimétricas.	A resolução de uma expressão obedece a ordem de avaliação definida pelo grau de precedência dos operadores.	Escrita correta da expressão.	Quanto vale x no final: int x=1, y=2, z=3; x= x * y - (z--);	x= -1
Agrupamento II (Ag II): soma secundária de classes.	Uso de parênteses altera a precedência dos operadores.	Como relacionar vários operadores numa expressão.	Diga se as expressões são equivalentes: a + b * c e a + (b * c) a * b + c e a * (b + c)	sim não
Agrupamento IV (Ag IV): multiplicação counívoca de classes.	A resolução de uma expressão que contenha tipos de dado distintos nem sempre é possível.	Uso de tipos de dados distintos na mesma expressão.	Quanto valem x, y, z, w no final: int x, y; int a = 7, b = 3; float z, w; x = a / b; y = a % b; z = a / b; w= z / b;	x= 2 y= 1 z= 2 w= 0.66
Lei da associatividade (LA3): expressa a possibilidade de se conseguir o mesmo resultado através de diversas operações alternativas.	Equivalência dos comandos: x ++ e x = x + 1, x -- e x = x - 1	Uso de operador unário e binário.	Corrija o código abaixo, se necessário: a ++ b; a ++; ++ a; a +; + a; a + b;	a + b; a ++; ++ a; a + b; b + a; a + b;

4º e 6º Passos: Agrupamentos x Exercícios sub-classe: Expressões condicionais

Leis do Agrupamento	Classe Expressões condicionais - Conceitos	Dificuldades dos novatos	Exercício Proposto	Possíveis Soluções
Reciprocidade (OpInter R): esta transformação permuta as afirmações e negações, sem alterar as conjunções e disjunções.	O resultado de uma negação é o oposto do estado anterior, o que é verdadeiro torna-se falso e vice-versa	Como usar o operador lógico de negação	As expressões abaixo são verdadeiras. Use o operador de negação para torná-las falsas: $(10/2 == 5);$ $((a < 3) \parallel (a > 7));$	$!(10/2 == 5);$ $!((a < 3) \parallel (a > 7));$
Correlatividade (OpInter C): esta transformação permuta as conjunções e disjunções, sem alterar as afirmações e negações.	Características da conjunção e da disjunção.	Uso do operador de disjunção ou de conjunção na construção da expressão condicional	Para $a = 1$ e $b = 2$, a expressão $(a > 0) \&\& (b > 5)$ é falsa. Altere a disjunção para conjunção e diga qual será o resultado	verdadeiro
Pensamento formal hipotético	Características da conjunção e da disjunção.	Converter uma frase em uma expressão condicional	Escrever uma expressão que traduza a frase: Para ser aprovado o aluno precisa ter nota de prova superior ou igual a cinco e nota de trabalho superior ou igual a sete.	$(nota \geq 5) \&\& (trab \geq 7)$
Pensamento formal hipotético	Características da conjunção e da disjunção.	Variação dos estados possíveis de uma expressão condicional	Dada a expressão: $(a == 6) \&\& (b != 6)$, definir valores para a e b que a tornem verdadeira	$a = 6,$ $b = 1$

Exercícios sobre Comandos

1º Passo: Dificuldades encontradas pelos novatos:

- 1) quando é preciso colocar chaves nos comandos;
- 2) como usar o while;
- 3) como usar o for;
- 4) quando usar o if e quando usar o for ;
- 5) quando usar o for e quando usar o while;
- 6) uso de comandos if encadeados;
- 7) uso de comandos for encadeados;
- 8) não utilização da indentação, dificultando a leitura do comando.

2º Passo: Características gerais da classe:

- Os comandos na linguagem C são:
 - estruturas de fluxo de controle, como if-else, switch, for, while, do-while, break, continue;
 - expressões, chamadas de funções, declarações, etc., quando seguidas de ponto-e-vírgula.
- Os comandos são terminados por ponto-e-vírgula.
- As chaves { e } são usadas para agruparem comandos num comando composto ou bloco.
- Não há necessidade de ponto-e-vírgula após a chave direita que termina um bloco.
- A ordem em que os comandos são organizados define a sequência em que a computação é feita, e corresponde ao fluxo de controle do programa.
- Erros no fluxo de controle são chamados de erros lógicos.

3º Passo: Características da Sub-classe: Repetição

- Os comandos de repetição são três: for, while e do-while.
- A sintaxe do comando while é :

```
while (expressão)
    comando;
```


a expressão é avaliada, se for verdadeira (diferente de zero), o comando é executado e a expressão é reavaliada. Este ciclo ou repetição prossegue até a expressão se tornar falsa (igual a zero), quando o fluxo de controle passa para o próximo comando da seqüência.

- a sintaxe do comando do- while é :

```
do  
    comando;  
while (expressão);
```

neste caso o comando é executado sempre, e só depois a expressão é avaliada. Se for verdadeira (diferente de zero), o comando é executado novamente e a expressão reavaliada. Este ciclo ou repetição prossegue até a expressão se tornar falsa (igual a zero), quando o fluxo de controle passa para o próximo comando da seqüência.

- a sintaxe do comando for é :

```
for (expressão1; expressão2; expressão3)  
    comando ;
```

a expressão1 é executada uma única vez no início, e tem a função de inicializar o contador usado no laço de repetição. Em seguida a expressão2 é avaliada, se for verdadeira (diferente de zero), o comando é executado. A expressão3 é então executada. A expressão 3 serve para atualizar o contador inicializado na expressão1. Em seguida a expressão2 é reavaliada. Este ciclo ou repetição: expressão3 – expressão2 prossegue até a expressão2 se tornar falsa (igual a zero), quando o fluxo de controle passa para o próximo comando da seqüência.

- as expressões 1 e 3 do for podem ser omitidas, neste caso o for se iguala ao while;

4º e 5º Passos: Agrupamentos x Exercícios Classe Comandos

Leis do Agrupamento	Classe Comandos – Conceitos	Dificuldades dos novatos	Exercício Proposto	Possíveis Soluções
Agrupamento I (Ag I): soma primária de classes ou encaixes simples ou inclusões, refere-se a compreensão de classe e sub-classe.	As chaves { e } são usadas para agruparem comandos num comando composto ou bloco.	Quando é preciso colocar chaves nos comandos.	Coloque as chaves quando necessário: if (a >= MAX) a= MAX; b= 0; else { a++; b++; } }	if (a >= MAX) { a= MAX; b= 0; } else { a++; b++; } }
Identidade (I) (OpInter I): é a transformação nula, não altera em nada a sentença sobre a qual incide.	Clareza na apresentação do código.	Não utilização da indentação, dificultando a leitura do código.	Identite o código: if (a <= b) { a--; b++; } else a++;	if (a <= b) { a--; b++; } else a++;
Não há.	Os comandos são terminados por ponto-e-vírgula.	Uso do ponto-e-vírgula final ao comando.	Corrija o código, se necessário: int a= 3, b=5 if (a < b) { a= a+b b= a-b }	int a= 3, b=5; if (a < b) { a= a+b; b= a-b; }

4º e 6º Passos: Agrupamentos x Exercícios sub-classe: Comandos de Repetição

Leis do Agrupamento	Classe Comandos de Repetição - Conceitos	Dificuldades dos novatos	Exercício Proposto	Possíveis Soluções
Agrupamento II (Ag II): soma secundária de classes, refere-se a compreensão de sub-classes complementares a uma classe.	Propriedades do comando for e do while.	Quando usar o while e quando usar o for.	Declare o vetor vet de inteiros com 10 posições, atribua números pares consecutivos ao vetor, iniciando por zero. Percorra o vetor vet, acima inicializado, até encontrar um número maior do que 4.	<pre>int vet [10]; int i; for (i=0; i< 10; i++) vet[i]= i*2; i=0 while (vet[i] <= 4) i++;</pre>
Agrupamento IV (Ag IV): multiplicação cunívoca de classes, refere-se a compreensão de uma estrutura com ramificações que se multiplicam, semelhante a estrutura de uma árvore.	Os comandos de repetição podem ser encadeados.	Uso de comandos for encadeados.	Complete o código, inicializando a matriz mat [2x5] de inteiros, com zeros int mat [2][5]; int i, j; for (; ;) mat[i][j]= 0;	<pre>int mat [2][5]; int i, j; for (i=0; i< 2; i++) for (j= 0; j < 5; j++) mat[i][j]= 0;</pre>

No item 5.1.14 - Protótipo dos Requisitos, será visto em detalhes como implementar um ambiente semi-dirigido segundo as teorias estudadas.

5.1.13 Obtendo Requisitos Funcionais e Não-Funcionais

Voltando aos casos de uso, e seus cenários, serão obtidos os *requisitos funcionais e os não funcionais* de cada cenário.

Os requisitos funcionais descrevem as especificações da funcionalidade, as ações, advindas do propósito principal do produto. Os requisitos funcionais podem dar origem a sub-requisitos.

Os requisitos não-funcionais são as propriedades ou qualidades que o sistema computacional deve conter que o tornam atrativo e próprio ao uso. Este tipo de requisito não define ou altera a funcionalidade do produto, mas confere a este uma característica que corresponde a impressão causada ao usuário pelo seu uso. Possui caracter subjetivo, e está relacionado à Qualidade em Software (capítulo 4, item 4.5), e aos recursos pedagógicos e tecnológicos (capítulo 2).

Os requisitos não-funcionais podem ser de duas categorias: globais aplicam-se a todos os casos de uso (cenários), ou locais, aplicam-se a um certo caso de uso.

Requisitos Não-Funcionais Globais:

- O sistema deve enviar mensagens objetivas e delicadas aos usuários, indicando uma forma alternativa de proceder, sempre que houver desentendimentos de comunicação.
- O sistema deve usar cores associadas às mensagens

A seguir cada cenário foi analisado na obtenção dos requisitos:

1º Cenário : Identificação do usuário

Informação: Dados do usuário (e)

Requisitos Funcionais:

- O sistema deve interagir de modo distinto a cada tipo de usuário.
 - O usuário pode ser professor ou aluno.

- O sistema deve identificar o usuário.
 - Novos alunos receberão uma ficha de inscrição, onde deve constar o seu nome, a sua turma, seu e-mail e o seu professor.
 - Os professores serão inscritos por meio de uma senha mestre do sistema.

2º Acompanhamento das estatísticas de aprendizado do aluno e da turma (professor)

Informação: Relatório de aproveitamento do aluno e da turma (s)

Requisitos Funcionais:

- O sistema deve registrar as principais ações de cada usuário.
- O sistema deve fornecer um relatório do aproveitamento do aluno.
 - O relatório de aproveitamento do aluno deve conter o tempo de uso do software, o número de exercícios feitos, o número de acertos e o de erros.
- O sistema deve fornecer um relatório do aproveitamento da turma.
 - O relatório de aproveitamento da turma deve conter a média e o desvio-padrão dos seguintes dados: tempo de uso do software, número de exercícios feitos, número de acertos e erros de cada exercício.

Requisitos Não-Funcionais:

- O sistema deve usar gráficos comparativos do aproveitamento do aluno com a média da turma
- O sistema deve usar cores na construção dos gráficos
- O tempo de resposta na elaboração do relatório não pode ser superior a 15 segundos.

3º Edição de exercícios (professor)

Informação: Novo exercício (e)

Requisitos Funcionais:

- O sistema deve conter um editor de exercícios.
 - O editor de exercícios deve conter o enunciado e as possíveis respostas.

4º Comunicação entre professores e alunos

Informação: Dúvidas (e) e Respostas (s)

Requisitos Funcionais:

- O sistema deve receber e enviar comunicações entre alunos e professores.
 - Um aluno pode mandar uma mensagem para seu professor.
 - Um aluno pode mandar uma mensagem para a lista de discussão.
 - Um professor pode mandar uma mensagem para seus alunos, usando ou não a lista de discussão.
 - Um professor pode mandar uma mensagem para um aluno.
- O sistema deve manter um histórico das mensagens enviadas, separadas por categorias.
- O sistema deve sinalizar a chegada da nova mensagem ao identificar o usuário.

Requisitos Não-Funcionais:

- As mensagens podem ser priorizadas pelo uso de cor

5º Atualização das referências bibliográficas

Informação: Referências Bibliográficas (e)

Requisitos Funcionais:

- O sistema deve conter as referências bibliográficas do curso.
 - As referências bibliográficas são compostas por livros, artigos, apostilas e links para sites da Internet.
- O sistema deve ter uma área para as referências bibliográficas feitas pelo professor e outra área para as referências feitas pelos alunos.
- A edição de uma referência deve ser feita através de um formulário.

Requisitos Não-Funcionais:

- As referências devem ser mostradas de modo uniforme e separadas por categorias.

6º Obtenção das notas de aula e dos exercícios propostos

Informação: Listagem das notas de aula e exercícios propostos(s)

Requisitos Funcionais:

- O sistema deve disponibilizar ao aluno as notas de aula e exercícios propostos pelo professor.
- Deve existir um formato padrão de apresentação para as notas de aula.

Requisitos Não-Funcionais:

- As notas de aula devem estar disponíveis no dia da aula.

7º Atividades práticas com exercícios

Informação: Diálogo com o tutor (e/s)

Requisitos Funcionais:

- O sistema deve interagir com o aluno num ambiente semi-dirigido de aprendizagem.
- O ambiente semi-dirigido é composto por um menu de botões, pela área de edição e pela área do enunciado do problema.
 - Os botões podem ser de quatro tipos:
 - a) botões com máscaras da sintaxe de um comando,
 - b) botões de inicialização de uma variável,
 - c) botões de inclusão de um operador aritmético, lógico ou relacional,
 - d) botão de execução.
 - A área de edição é composta por três áreas:
 - a) Cabeçalho da função
 - b) Declaração de variáveis
 - c) Corpo da função
- Antes de executar o código, o aluno deve prever a resposta do sistema a uma entrada específica, mostrando seu entendimento sobre o problema.
- Existe uma sequência pré-definida de apresentação de exercícios.

Requisitos Não-Funcionais:

- A cor adotada por uma classe deve ser a mesma usada na edição do código.

- As áreas do ambiente de trabalho semi-dirigido: enunciado do problema, cabeçalho, declaração de variáveis e código, devem ser distintas através de um fundo colorido suave.

8º Obtenção de referências da linguagem C

Informação: Referência da Linguagem C (s)

Requisitos Funcionais:

- O sistema deve dispor de uma ajuda on-line sobre os tópicos ensinados.
- A referência da linguagem pode ser solicitada pelo aluno.
- A referência da linguagem pode ser mostrada ao aluno em caso de sucessivos erros.
- As referências devem ser à linguagem e aos erros mais comuns.
 - A referência à linguagem conterá uma breve explicação sobre o tema, seguida de exemplos.
 - A referência aos erros conterá uma breve explicação sobre um erro freqüente , seguida do código com erro e o correto, permitindo ao aluno compará-los.

Requisitos Não-Funcionais:

- Antes de mostrar a referência ao aluno que errou, o sistema deve pedir sua permissão.

9º Obtenção de referências bibliográficas

Informação: Relatório de Referências Bibliográficas (s)

Requisitos Funcionais:

- O sistema deve disponibilizar as referências bibliográficas do professor.
- O sistema deve disponibilizar as referências bibliográficas dos alunos.

10º Obtenção de estatística de seu desempenho (aluno)

Informação: Relatório de Aproveitamento (s)

Requisitos Funcionais:

- O sistema mostra ao aluno seu aproveitamento do curso, em número absoluto de ações efetuadas e comparativamente à média do grupo.
- O aproveitamento do aluno é medido pelos critérios: tempo de utilização do sistema, número de exercícios trabalhados, número de exercícios resolvidos.
- Não é permitido um aluno obter dados de aproveitamento de outros alunos.

Requisitos Não-Funcionais:

- O relatório deve ser apresentado em uma única folha.

11º Obtenção de listagem dos exercícios resolvidos

Informação: Lista de Exercícios Resolvidos (e/s)

Requisitos Funcionais:

- O sistema disponibiliza ao aluno a resolução dos exercícios trabalhados.

5.1.14 Protótipo dos Requisitos

- I. Neste etapa do levantamento de requisitos, já foram obtidas informações suficientes para a elaboração de um protótipo do sistema. O protótipo elaborado consiste na definição e descrição das telas, no entanto não possui funcionalidade.

A partir dos casos de uso, dividi a operação do sistema em cinco áreas principais:

- Exercícios: é aqui que o aluno efetivamente vai exercitar-se na matéria através de atividades práticas semi-dirigidas. Incorporado aos exercícios está um guia de referência da linguagem C. Nesta área haverá também um editor de exercícios que permitirá aos professores a geração de material específico às dificuldades de uma turma.
- Professor: nesta área estão as notas de aula (quando houver), as notas das provas e das listas, identificação e horários de monitoria, e os exercícios propostos. Este espaço possui uma moldura, mas é o próprio professor que define o conteúdo;
- Aproveitamento: é mantido um histórico de cada aluno, feito a partir de um rastreamento da interação do aluno com o projeto e estatísticas de aproveitamento da turma.

- Referências Bibliográficas: contém referências a livros e artigos, e links para sites ;
- Comunicações: é o local onde alunos e professores trocam mensagens, possui um quadro de avisos, lista de discussão e os endereços eletrônicos dos alunos. Possui conexão com o correio eletrônico;

A Figura 8 mostra uma primeira versão para a tela principal e a tela de exercícios do software educativo.

Figura 8 - Telas do protótipo

O protótipo desenvolvido não possui nenhuma funcionalidade, sendo desenvolvido em papel.

Com a definição das áreas do sistema, estabeleci a rede de navegação, baseada em hipertexto (Figura 3).

Os exercícios propostos irão abordar individualmente o conteúdo de cada uma das classes identificadas (item 5.1.12.2).

Figura 9 - Estrutura de navegação do protótipo

Ambiente Semi-Dirigido

O ambiente de trabalho semi-dirigido possui os principais elementos da linguagem C: na borda superior estão os botões de comandos, e os botões de operadores aritméticos, relacionais e lógicos;

- na lateral esquerda estão os botões de declaração e/ou inicialização de variáveis e declaração de constantes;
- na área central temos uma sub-divisão: acima está o enunciado do exercício e abaixo fica a área de edição.
- na área de edição, o espaço está sub-dividido em 3 partes: cabeçalho, declaração de variáveis e comandos.

AMBIENTE DE TRABALHO SEMI-DIRIGIDO - COMANDOS

IF THEN ELSE SWITCH FOR WHILE DO WHILE

Operadores Aritméticos: +, -, *, /, %, ++, --

Operadores Relacionais: >, <, >=, <=, ==, !=

Operadores Lógicos: &&, ||, !

VARIÁVEIS

Enunciado do problema

Cabeçalho

Declaração de variáveis

Comandos

CONSTANTES

EXECUTA

Figura 10 - Protótipo do ambiente de trabalho semi-dirigido

Um problema é apresentado ao aluno, que precisa:

- 1) identificar quais são os elementos da linguagem que serão usados;

- 2) seleccionar o botão;
- 3) completar os dados quando necessário;
- 4) definir valores de entrada e os respectivos valores de saída;
- 5) executar o trecho de código;
- 6) observar o resultado obtido com o esperado;
- 7) voltar ao passo 1 caso o resultado obtido seja diferente do esperado;
- 8) uma resposta correta é mostrada ao aluno, caso seja por ele solicitado.

Interação com o Ambiente de Trabalho Semi-Dirigido

O termo semi-dirigido é usado no sentido de propor desafios ao aprendiz, mas não guiá-lo ou induzi-lo na busca de uma solução, como ocorre num teste de múltipla escolha, deixá-lo

Figura 11 - Declaração de variável

explorar as várias possibilidades que existem no ambiente, simular hipóteses e checar os resultados, este é um modo de agir sobre os novos conceitos. O ambiente semi-dirigido procura prover meios ao aluno de adquirir o conhecimento sobre programação.

Uma pequena mostra da interação pode ser vista nas figuras: a declaração de variáveis é mostrada na seqüência da Figura 11, e a inclusão do comando IF na Figura 12.

Figura 12 - Inserção do comando IF

Os demais comandos: switch, do-while e for seguem este mesmo estilo de interação.

Mas como usar os exercícios elaborados, no item Elaboração de Exercícios (5.2.12.2), no protótipo do ambiente de trabalho semi-dirigido ?

O enunciado do problema é apresentado ao aluno. Alguns dos exercícios propostos fornecem um código inicial, para ser completado ou corrigido. Neste caso este código também será incluído, caso haja variáveis, suas caixas de declaração correspondentes estarão na área de variáveis.

A fim de ilustrar esta passagem um exercício propostos sobre variáveis será transposto para o protótipo (Figura 13). Podemos observar que neste exercício o aluno teve que se preocupar apenas com o tipo de dado da nova variável, as áreas de cabeçalho e comandos não são necessários na resolução e ficaram em branco.

The figure shows two side-by-side screenshots of a software development environment titled "AMBIENTE DE TRABALHO SEMI-DIREITO - COMANDOS".

Left Panel (Proposed Exercise):

- COMANDOS:** IF THEN ELSE, SWITCH, FOR, WHILE, DO WHILE.
- Operadores:** Three tabs: Operadores Aritméticos, Operadores Relacionais, Operadores Lógicos.
- VARIÁVEIS:** A section with the instruction "Declare a variável soma, que guardará o resultado de operações entre nota1 e nota2:". It contains input fields for "nota1" and "nota2", and a larger field for "soma". Below these are labels "float nota1;" and "float nota2;".
- CONSTANTES:** A section with a large empty text area.
- EXECUTA:** A button at the bottom.

Right Panel (Resolution):

- COMANDOS:** Same as the left panel.
- Operadores:** Same as the left panel.
- VARIÁVEIS:** Similar to the left panel, but with an additional input field for "soma" and a label "float soma;".
- CONSTANTES:** Same as the left panel.
- EXECUTA:** Same as the left panel.

Figura 13 - Exercício proposto (esquerda) e sua resolução (direita)

O botão de execução compila o código, identificando erros sintáticos, e caso não haja erros, executa o código, atualizando o valor das variáveis. A resposta do aluno é comparada com a resposta correta pré-definida. As diferenças, se houver, são mostradas sublinhadas ao aluno, que deve observá-las atentando para os possíveis erros. Nesta etapa o aluno pode solicitar o Guia de referência da linguagem.

Após a 2ª tentativa sem êxito, uma das possíveis respostas corretas é apresentada ao aluno através de um click duplo no botão Executa, evitando que o aluno com maiores dificuldades desmotive-se, desistindo de resolver o problema.

A avaliação neste ambiente restringe-se ao número de tentativas do aluno em tentar resolver o problema e ao tempo gasto, e se chegou a uma solução correta ou não. O aluno pode

comparar seu resultado com o da resposta fornecida e com o resultado dos colegas. Não é proposta deste software educativo desenvolver um sistema de avaliação mais abrangente. Esta limitação resulta numa sequencialização padrão de apresentação de exercícios, pré-definida, embora seja possível a escolha específica de um exercício.

Este protótipo, no entanto não foi mostrado aos stakeholders, o que seria o próximo passo na obtenção de um feedback sobre sua operacionalidade.

5.1.15 Portal de Qualidade

O Portal de Qualidade aplica-se aos requisitos obtidos, realizando uma série de testes, que permitem testar conflitos e verificar se o requisito atende às especificações do sistema.

O Portal testa individualmente cada requisito em relação a: completude, rastreabilidade, consistência, relevância, corretude, ambigüidade, viabilidade, e fora do escopo do produto. Os requisitos são testados contra as restrições globais do sistema, e é feito um cruzamento requisito a requisito verificando incompatibilidades.

Em geral são verificados conflitos entre requisitos solicitados por diferentes stakeholders. Neste estudo os stakeholders atuaram indiretamente, pois suas necessidades foram buscadas através da lista de discussão do Curso C e da teoria Construtivista, passando sempre por uma análise prévia de como seriam implementadas. Com a confecção do protótipo pudemos observar que a inexistência de conflitos entre os requisitos, assegurando a passagem pelo Portal de Qualidade aos requisitos funcionais e não funcionais identificados.

5.2 Documento de Requisitos

O Documento de Requisitos de Software tem por objetivo registrar a análise de requisitos de modo sistemático, inserindo o desenvolvedor do software no contexto apropriado, definindo claramente o que o software deve fazer e delimitando seu escopo. Por outro lado, este documento presta-se também em assegurar ao cliente que suas necessidades serão atendidas; registrando as especificações que o software deve atender, como em um contrato.

No capítulo 4 (item 4.2.6) foram descritas as partes que este documento deve conter: descrição do contexto do sistema, descrição dos requisitos funcionais e não funcionais,

especificação dos requisitos, etc. Toda esta informação foi vista ao longo deste capítulo, durante o desenvolvimento do Método Volere, sendo desnecessária e imprópria sua repetição, a qual levaria o leitor a desinteressar-se. Por outro lado não poderíamos optar por apresentar o Documento de Requisitos ao invés do desenvolvimento da análise. A análise do problema trás consigo uma série de considerações e ponderações que no Documento ficam restritas aos resultados.

5.3 Conclusões deste capítulo

Neste capítulo foi desenvolvida a Análise de Requisitos do Software educacional de ensino da programação. Esta análise baseou-se nos estudos feitos nos capítulos anteriores. A fundamentação teórica teve suma importância no processo de avaliação das opções, feitas ao longo desta análise. Esta base teórica norteou o encaminhamento da análise obtendo resultados, se não comprovados, pelo menos justificados.

Obteve-se como resultado a especificação de um software educacional interativo, a qual espera-se propiciar ao aluno meios de explorar o conteúdo em aprendizado, orientando-o somente quando for necessário.

O desenvolvimento do protótipo foi essencial como complementação, levando a observação de situações que não poderiam ser previstas por outro meio.

No próximo capítulo faremos uma breve revisão sobre as conclusões a que este estudo chegou, bem como de suas limitações.

Capítulo 6

Conclusões sobre este trabalho

Neste capítulo é feita uma retrospectiva desta dissertação, percorreremos brevemente os capítulos iniciais procurando responder criticamente às perguntas feitas no início deste estudo e apresentar os resultados obtidos.

6.1 O Processo de trabalho e as opções feitas

No capítulo 1 o problema a ser estudado foi definido, sendo identificadas três questões fundamentais sobre o ensino da programação (Du Boulay e O'Shea, 1981).

Como o software proposto atende a estes três problemas?

I. Quais são as dificuldades encontradas pelos novatos ao aprender a programar?

O ato de programar pode ser dividido em três etapas:

- 1) planejamento,
- 2) codificação,
- 3) testes.

Quanto ao planejamento: atende parcialmente, pois o software educativo não está capacitado a tratar um programa completo, mas a partes deste, sendo pedido ao aluno para prever a resposta de um pedaço do código antes de sua execução.

Quanto à codificação: atende plenamente, já que identifica erros sintáticos e semânticos, devido aos exercícios elaborados trabalharem cada uma das classes identificadas no domínio da programação segundo a teoria piagetiana.

Quanto aos testes: atende parcialmente, pois apenas a parte codificada é testada. O resultado é avaliado pelo programa e pelo próprio aluno.

II. Quais são as características que uma linguagem de programação deve ter que a tornem mais apropriada a ser a 1ª. linguagem ensinada?

Segundo McCracken (In:Martin,1996,cap2) o ambiente de programação deve proporcionar ajuda ao desenvolvedor, sendo tão importante quanto a própria linguagem. Um ambiente de programação deve ter o verificador de erros sintáticos integrado a um bom editor de texto.

Apesar da linguagem C não ser considerada a linguagem mais apropriada como 1ª linguagem, é adotada no IME, conforme justificado pelos professores, no entanto este estudo aponta para a necessidade de ser escolhido com mais critério o ambiente de programação.

III. Qual é a natureza da habilidade a ser ensinada e qual o melhor modo de ensinar?

Estudamos no capítulo 2 as duas principais teorias propostas pela Psicologia Educacional: o behaviorismo e o cognitivismo. Para melhor entender o processo mental envolvido na aprendizagem buscamos auxílio na teoria de Piaget, o qual estudou o *desenvolvimento intelectual*, procurando determinar as diversas fases que o compõem e suas características marcantes; essencialmente preocupado com o processo de busca do conhecimento, e não apenas com seu resultado.

A natureza da habilidade a ser ensinada é o próprio processo mental, pois programar é uma atividade essencialmente abstrata, que usa símbolos e suas combinações para chegar a um resultado. Cada problema proposto leva o aluno a percorrer os processos de busca,

Figura 1 - Teoria do desenvolvimento intelectual de Piaget

acomodação e assimilação (Figura 1), sendo assim nada mais propício que usar a teoria de desenvolvimento intelectual de Piaget.

Ainda no capítulo 2 vimos que um software educativo, ao apresentar uma idéia nova ao aluno – *um elemento desconhecido e externo* – para torná-lo *um elemento conhecido e interno*, deve portanto propiciar condições de testar de todas as maneiras este novo elemento, explorado-o quanto às suas características, a que classe pertence, quais são suas propriedades, e como se relaciona com os demais elementos presentes em sua estrutura mental pré-conhecida. Somente desta forma este novo elemento será conectado aos demais elementos da rede mental, tornando-se um novo nó desta rede dinâmica. Um software educativo deve levar o aluno a refletir sobre os resultados de suas simulações. As simulações feitas no software educativo são as ações necessárias à internalização do processo, e neste ponto a ferramenta computador está apta a prover esta situação de aprendizagem.

Dentro desta abordagem teórica, toda aprendizagem é ativa. É o sujeito que parte em busca do conhecimento daquilo que acha que falta. O conhecimento não se reduz a uma transmissão de informação de fora para dentro dada pelo professor, mas supõe uma ação mental do sujeito que aprende.

Para assimilar o conhecimento, o sujeito deve fazer um esforço interno para relacionar o novo objeto do conhecimento ao conhecimento já adquirido. Em suma, para que haja ensino/aprendizado é necessário que a nova informação seja organizada pelo sujeito (nível formal do aluno) e que ela seja significativa para os registros internos anteriores do aluno (nível de conteúdo semântico/ tema).

A verbalização do processo em seus dois sentidos (idas e vindas), assim como do que já aconteceu e do que vai acontecer, com o registro mental dos êxitos e fracassos obtidos, leva do mero fazer por ensaio e erro ao compreender as seqüências e a utilizá-las de forma intencional, refletida e ágil. As tentativas impulsivas, os toques irrefletidos vão pouco a pouco sendo substituídos por uma execução planejada, organizada, com grande aproveitamento de indícios, utilização da memória, exclusão de movimentos desnecessários e crescente coordenação de ações.

O software educacional propõe um ambiente semi-dirigido de exercícios visando propiciar ao aluno, um local onde ele possa testar as hipóteses sobre o novo elemento, agindo e experimentando suas propriedades.

Em relação aos recursos tecnológicos, além da multimídia e do hipertexto, o ponto inovador visto foi a introdução da IHC nos currículos dos cursos de Ciência da Computação, confirmando a utilização do computador pela sociedade em geral não mais como uma máquina capaz de realizar cálculos complexos, mas antes como um veículo de comunicação de idéias. Neste novo contexto a palavra chave é usabilidade, a qual identifica as heurísticas a serem seguidas para criação de uma interface interativa e amigável.

No capítulo 3 acompanhamos a evolução dos softwares educacionais, dos primeiros CAI baseados nas instruções programadas de Skinner aos ITS, ‘o protótipo do professor ideal’ e totalmente automatizados, mas ainda em processo de constituição, devido às dificuldades computacionais. Os softwares educacionais baseiam-se numa individualização do ensino, seja CAI ou ITS, individualização que visa respeitar as características próprias de cada aluno, como seu ritmo, ou sua maior ou menor facilidade de compreender um conceito; no entanto, vimos que mais recentemente outro aspecto não menos importante está sendo considerado: a colaboração entre os alunos. Deste modo ocorrem dois movimentos complementares o da individualização numa primeira etapa e o da colaboração numa segunda etapa, que atuam ciclicamente, ao longo do processo de aprendizagem.

Do campo específico do ensino da programação, foram destacados sete softwares, e pudemos observar nestes softwares, como ainda está deficiente o desenvolvimento de softwares educacionais, no sentido de atender às necessidades e expectativas dos usuários, se tomarmos como base de comparação outras categorias de características igualmente interativas como editores de texto, ou softwares de navegação em hipertexto -‘browsers’.

Vimos que, infelizmente após tantos anos, os ITS ainda estão em estágio embrionário, não sendo efetivamente usados nas Universidades ou Escolas. O pouco sucesso dos ITS talvez esteja relacionado à alguns fatores, observados por mim, neste estudo:

- *dificuldades computacionais relativas ao tempo de resposta das linguagens declarativas;*
- *dificuldades computacionais relativas à comunicação humano-computador;*

- *dificuldades computacionais em compor e atualizar a base de conhecimento sobre o assunto estudado.*

Na evolução do software educacional observa-se a importância da comunicação deste com o aluno, ponto chave da relação ensino-aprendizagem. A evolução dos softwares mostra que o alvo continua sendo o modelo do tutor humano, um tutor personalizado frente ao aluno, com alto grau de interatividade, ideal buscado pelos ITS, os quais ainda estão em estágio de pré-paradigma. Para que suas metas venham a se concretizar, é necessária a evolução, em suas várias formas, do relacionamento do homem com o computador.

As novas disciplinas da Ciência da Computação mostram esta tendência, como o IHC que preocupam-se exclusivamente com o desenvolvimento da interatividade, visando uma comunicação mais abrangente em seus diversos níveis.

Atualmente os cursos oferecidos virtualmente lançam mão dos recursos advindos das tecnologias de comunicação e da Internet, proporcionando uma alternativa de aprendizado ao ensino tradicional, são os cursos de educação à distância,. Pode-se destacar aqui a importância da aprendizagem cooperativa, baseada nas idéias de Vygotsky, presente em muitos cursos à distância.

Na área específica do ensino da programação, observa-se a carência de softwares específicos, e o alto interesse que há pelo assunto – havia 2500 inscritos no curso de C da UFMG. O Projeto MacMultimídia teve um bom retorno também.

A partir dos estudos feitos constatamos que na prática, não há uma divisão clara entre os paradigmas: o tradicional CAI e o ITS baseado em Inteligência Artificial, a proposta apresentada nesta dissertação busca justamente conciliar estes dois polos: o possível e o desejável, sendo assim apesar de não serem usadas técnicas de IA, o sistema proposto quer deixar o aluno ser o guia de seu aprendizado, através um ambiente interativo semi-dirigido, onde as atividades são propostas permitindo ao aluno diversos modos de realizá-la.

No capítulo 4 vimos como conduzir a Análise de Requisitos, suas principais etapas e os aspectos mais relevantes deste processo.

No capítulo 5 foi apresentada a Análise de Requisitos para o software educacional de ensino da programação. A metodologia de análise selecionada foi o Método Volere, pois não encontramos uma técnica específica ao domínio tão peculiar do software educativo. O Método Volere possui uma seqüência bem definida e criteriosa de etapas e baseia-se nos casos de uso, tendo como finalidade conduzir ao entendimento de forma mais abrangente do domínio estudado. A abordagem por *casos de uso* na análise de requisitos, conduziu à obtenção dos requisitos para cada um dos casos de uso. É uma forma de se dividir o sistema em partes menores, de modo eficiente, pois os casos de uso representam cada um dos serviços a ser feito pelo sistema em estudo. Os requisitos obtidos desta forma referem-se ao ator, ou seja o que o sistema deve fazer para permitir que haja uma interação satisfatória com o ator.

O processo de obtenção de requisitos, seguiu os passos previstos pelo Volere, mas não foram estes suficientes, sendo necessário a inclusão de etapas adicionais:

- I. O Diagrama de Contexto mostrou os sistemas adjacentes e sua interação com o sistema proposto através dos eventos;
 - Foi preciso incluir um diagrama extra, mostrando os sistemas adjacentes necessários durante a confecção do sistema, mas que não interagem com o sistema pronto;
- II. Os eventos foram identificados e listados;
- III. Para cada evento foi descrito um cenário, ou seja como o sistema reage em resposta ao evento;
 - O cenário *Atividades práticas com exercícios* necessitou de todo um estudo detalhado de como seria desenvolvido, pois é uma nova proposta de interação e de condução da aprendizagem. Este estudo envolveu duas etapas: 1) Elaboração de exercícios baseado nas leis dos Agrupamentos da teoria piagetiana e 2) Elaboração de um ambiente de trabalho semi-dirigido de acordo com o construtivismo;
 - Para o estudo deste cenário foram feitos questionários, que se mostraram inadequados, e buscados os erros de programação mais freqüentes obtidos a partir de um curso de C à distância;

- IV. Foram definidos os requisitos funcionais e os não-funcionais;
- V. Para compreender melhor como seria o software educativo foi desenvolvido um protótipo, consistido pela definição e descrição das telas, mas sem funcionalidade.

Para elaborar os exercícios foi feita a divisão do conteúdo a ser aprendido em classes, visando propiciar a compreensão de:

- *uma classe,*
- *relações existentes entre as classes,*
- *possíveis transformações sobre os relacionamentos.*

Sendo assim, o sistema aqui proposto tem um enfoque distinto dos demais softwares educacionais voltados ao ensino da computação, onde desde a primeira aula, é vista a programação de uma função com todos os elementos que a compõe. Tal aproximação dificulta a classificação pelo aluno dos vários elementos envolvidos. O que foi proposto é dividir por partes os elementos de um programa, trabalhando-os da forma mais concreta possível, através de simulações. Para esta divisão, foram definidas classes e sub-classes sobre o conteúdo do domínio a ser estudado.

A proposta de elaboração de exercícios segue os passos:

- *1º Passo: as dificuldades mais frequentes encontradas pelos alunos;*
- *2º Passo: as características gerais da classe;*
- *3º Passo: as particularidades de cada sub-classe,*
- *4º Passo: os agrupamentos que promovem um melhor entendimento da classe,*
- *5º Passo: os exercícios para a classe;*
- *6º Passo: os exercícios para a sub-classe.*

O cruzamento das leis da teoria piagetiana com as classes identificadas no domínio da programação, não foi feito de forma exaustiva, sendo destacados os cruzamentos mais relevantes a fim de mostrar como o processo pode ser conduzido.

O Protótipo do software foi desenvolvido a partir da definição das áreas do sistema, sendo definida a rede de navegação, baseada em hipertexto. Os exercícios propostos abordam individualmente o conteúdo de cada uma das classes identificadas.

Para realizar os exercícios foi proposto um ambiente de trabalho semi-dirigido. O termo semi-dirigido é usado no sentido de propor desafios ao aprendiz, mas não guiá-lo ou induzi-lo na busca de uma solução, como ocorre num teste de múltipla escolha, deixá-lo explorar as várias possibilidades que existem no ambiente, simular hipóteses e checar os resultados, este é um modo de agir sobre os novos conceitos. O ambiente semi-dirigido procura prover meios ao aluno de adquirir o conhecimento sobre programação.

Esta recapitulação procurou identificar os trechos marcantes do desenvolvimento deste estudo, oferecendo um sumário dos capítulos anteriores. Vejamos agora quais são as limitações deste trabalho.

6.2 Limitações

No item anterior tratamos de assuntos pertinentes à área circunscrita por este estudo, cabe-nos agora destacar os assuntos, mesmo que vizinhos ou próximos, não pertencentes a esta delimitação.

No levantamento das dúvidas do curso de C da UFMG, pudemos identificar algumas questões não são tratadas pelo software educacional:

- como configurar o compilador;
- como usar o compilador;
- como configurar o sistema operacional;
- como usar o teclado;
- como depurar o programa.

Estas dúvidas dizem respeito aos demais elementos envolvidos no sistema de ensino, identificados pelo Diagrama do Modelo Proposto (capítulo 5, item 5.2.6), como compilador, sistema operacional, mas que não são tratados pelo software educativo.

Não faz parte deste software ensinar programas completos, mas sim partes de um programa, o que é sem dúvida uma grande limitação, mas é o que é possível de ser feito, atualmente.

6.3 Avaliação Crítica – Resultado Científico

Ao estudar o problema do software educacional de ensino da programação, encontrei duas linhas bem distintas de conduzi-lo: uma procurava através de ITS automatizar totalmente aprendizagem excluindo o papel do professor no processo de ensino, e a outra linha usava o computador como veículo de comunicação, deixando o aprendizado por conta do próprio aluno, que resolvia suas dúvidas através de e-mails.

Procurei então viabilizar um sistema que dispusesse de um ambiente de aprendizado individualizado, permitindo ao aluno praticar exercícios, mas que fosse integrado à aula, tendo o professor papel fundamental no processo de aprendizagem.

A Psicologia Educacional e sua teoria construtivista proveram os recursos necessários que embasaram as propostas feitas tanto para elaboração de exercícios, como na construção do ambiente de trabalho semi-dirigido.

Estas propostas basearam-se também na IHC e no hipertexto, visando criar um sistema computacional agradável ao uso, cuja interação seja natural ao usuário.

O protótipo, no entanto, não foi testado pelos stakeholders, o que traria sem dúvidas benefícios à análise, a partir de sugestões e críticas levantadas, no entanto ocasionaria uma nova rodada em todas as etapas da Análise de Requisitos.

Anexo I

Obtenção de Requisitos - Questionários e Entrevistas

A obtenção de requisitos é feita observando-se como o trabalho é feito. O software educativo irá atuar fora do horário de aula, quando os alunos estiverem resolvendo os exercícios propostos pelo professor, sendo assim o trabalho consiste em resolver problemas de programação.

Objetivo: Observar o ambiente de trabalho atual dos alunos, fora e dentro do laboratório.

Contexto do trabalho: integrar a aula teórica às atividades práticas de programação.

Metodologia: Os alunos são observados em duas situações: fora e dentro do laboratório. Numa sala é dado um problema aos alunos, e observada a resolução. Os alunos são divididos em grupos de 2; através do diálogo as etapas de desenvolvimento são melhor observadas pelo pesquisador. Após a resolução ser feita em papel e lápis os alunos lêem as observações feitas, comentando-as, e completando-as. A 2ª situação ocorre no laboratório, onde os alunos vão usar o compilador, testando o problema resolvido anteriormente. No final, os alunos respondem um questionário sobre sua experiência anterior no uso de computadores.

Problema proposto: Dado n calcular a soma dos n primeiros números naturais.

(Dado n natural, calcular $n!$)

(Dados x inteiro e n natural, calcular x elevado a n)

1ª Fase – Na sala

Tempo: 30 minutos.

Observações feitas pelo pesquisador durante o trabalho dos alunos:

1. Os alunos elaboraram uma solução em pseudo-código?
☐ sim

- ☐ não

- 2. Os alunos começaram o problema escrevendo:
 - ☐ protótipo
 - ☐ declaração de variáveis
 - ☐ corpo da função

- 3. Os alunos citaram frases ditas pelo professor durante a aula?
 - ☐ sim
 - ☐ não

- 4. Os alunos consultaram o caderno procurando exemplos?
 - ☐ sim
 - ☐ não

- 5. Os alunos consultaram livros de programação?
 - ☐ sim
 - ☐ não

- 6. Os alunos realizaram uma simulação ao término?
 - ☐ sim
 - ☐ não

- 7. Houve divergências significativas entre os alunos?
 - ☐ sim
 - ☐ não

Espaço para os comentários do grupo

2ª Fase – No laboratório

Tempo: 45 minutos.

Observações feitas pelo pesquisador durante o trabalho dos alunos:

1. Como os alunos interagiram com o S.O.?

- ☐ login na rede
 - ☐ sem ajuda
 - ☐ com ajuda
 - ☐ iniciar o compilador C
 - ☐ sem ajuda
 - ☐ com ajuda
2. Uso do Compilador C.
- ☐ usou com segurança
 - ☐ precisa de alguma ajuda
 - ☐ não sabia usar
3. Digitação do programa. Os alunos colocaram comentários?
- ☐ sim
 - ☐ não
4. Erros de sintaxe. Compreensão das mensagens do compilador?
- ☐ leu e entendeu o que deveria ser alterado
 - ☐ leu e não entendeu a mensagem
 - ☐ não leu
5. Perseverança.
- ☐ o aluno tirou todos os erros de sintaxe
 - ☐ desistiu de arrumar o programa
6. Dúvidas. Como os alunos agiram para corrigir o código?
- ☐ consultou as notas de aula
 - ☐ consultou o livro de referência da linguagem
 - ☐ consultou os colegas
 - ☐ consultou o professor ou monitor
7. Erros de lógica. Estando o programa compilado, os alunos:
- ☐ acham que a programação está pronta e dão por encerrada a atividade
 - ☐ roda o programa, testando-o
8. Simulação. Os alunos sabem simular o programa?

- ☐ sim
- ☐ não

9. Os alunos sabem verificar se a saída do programa está certa? Como é feito?

Questionário Final - Individual

1. Você se sente bem frente a um computador?
 - ☐ sempre
 - ☐ às vezes
 - ☐ nunca
2. Em geral, quanto tempo você usa o computador?
 - ☐ mais de 2 horas por dia
 - ☐ em torno de 1 hora por dia
 - ☐ em torno de 1 hora por semana
 - ☐ raramente
3. Qual o tipo de computador que você usa?
 - ☐ PC (Personal Computer) com Windows
 - ☐ PC com Linux
 - ☐ PC conectado a uma rede local (Windows NT, Novell)
 - ☐ PC com Linux conectado a uma rede local
 - ☐ MAC (Machintosh)
 - ☐ Outro:
 - ☐ Nenhum
4. O que leva você a usar o computador?
 - ☐ distração, divertimento
 - ☐ uso da Internet. Especifique:
 - ☐ edição de textos
 - ☐ planilhas
 - ☐ estudo com software educativo
 - ☐ outros:
5. Marque os programas que você sabe usar e o nível:

☐ Word	☐ básico ☐ avançado
☐ Excel	☐ básico ☐ avançado
☐ Jogos em geral	☐ básico ☐ avançado
☐ Netscape ou Internet Explorer	☐ básico ☐ avançado
☐ ICQ	☐ básico ☐ avançado
☐ outros:	

6. Como você se sente ao usar este laboratório?
 - ☐ confiante
 - ☐ pouco a vontade
 - ☐ sinto-me totalmente perdido
 - ☐ tenho vergonha de errar ou cometer erros
7. Qual foi sua maior dificuldade no laboratório?
8. Você já teve algum contato com programação? Se sim com qual programa e o que fez?
9. Qual foi sua maior dificuldade em seus primeiros programas, aqui na faculdade?

Anexo II

Levantamento de Dúvidas

Do curso de C da UFMG (capítulo3, item 3.2.6), percorri a lista de discussão¹, da edição do 2º Semestre de 2000, selecionando dúvidas e classificando-as por tipo. A lista de discussão contém mais de 900 mensagens, destas percorri as primeiras 150, obtendo no entanto dúvidas bem diversificadas. Esta fonte de dúvidas e erros será usada na preparação dos exercícios.

Observe que os textos a seguir, são trechos de-mail, significantes ao nosso estudo. Os textos foram mantidos em sua forma original com erros ortográficos e gramaticais, note que alguns alunos expressam-se muito mal. Bom mas este é um assunto para outro estudo...

Dúvidas sobre variáveis

Resumo:

- 1) como é a alocação de memória no union
- 2) uso de uma variável antes de declará-la
- 3) ler o valor de uma variável não inicializada
- 4) utilização de número inteiros acima do limite
- 5) expressões aritméticas com tipos distintos
- 6) uso de acentos ao nomear uma variável
- 7) variável extern
- 8) variável de ambiente (global)
- 9) uso dos modificadores de tipo: long, short

¹ <http://ead1.eee.ufmg.br/cursos/C/lista.html>

10) uso da vírgula como separador decimal

11) acesso a elementos de uma string

Dúvidas Originais

O QUE EU ESTOU FAZENDO ERRADO NESSE EXERCICIO O PUTS NAO FUNCIONA APARECE UMA LETRAS ESQUISITAS...

```
#include <stdio.h> /* funcoes de entrada e saidas de dados */

int main()
{
    char string1[50];
    char string2[80];
    printf("\n\n Entre com o seu nome: ");
    gets(string1);
    printf("\n\n O nome é: %s", string1);
    printf("\n\n Digite uma frase: %s\n", string2);
    Uai! Onde vc leu string2? É claro que é lixo. Teria q ter um gets antes do
    puts... Putz! :-)

    puts(string2);
    printf("\n\n A Frase: %s\n", string2);
    return 0;
}
```

Minha dúvida é a seguinte: como Union em C aloca memória? Pelo que entendi, ela aloca memória tomando como base o maior dos tipos, certo? Se for assim, como será alocada memória para os outros, ou seja, os outros serão compactados, ou terão, outras áreas de mesmo tamanho do tipo maior? Porque se for de acordo com essa última informação, acho que não haverá economia de memória. Por favor me tirem esta dúvida, pois no nosso laboratório trabalhamos muito com otimização da memória. Obrigado.

Rodei o programa com float e deu certo. Agora fiquei com uma dúvida: e se a gente quiser utilizar variáveis inteiras tão grande, o que fazemos, choramos ? :-)

Foi muito bem observado sua colocação sobre o código do long int. Agora repare isso:

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    double y, x = 1000000000;
```

```
    printf("O valor de x é: %g\n", x);
```

```
    x *= 10;
```

Aqui voce fez uma multiplicação de x (que é um double) por um inteiro. A operação é efetuada utilizando doubles.

```
    printf("O valor de x é: %g\n", x);
```

```
    y = (double)(1000000000 * 10);
```

Aqui voce fez a operação de um long int por um int e depois converteu o resultado para double. Ou seja, a

operacao em si e' feita usando longs ints, o resultado extrapola os limites dos longs ints e so depois e' convertido em double.

```
printf("O valor de y e: %g\n", y);
return 0;
}
```

Ou seja, como vc colocou deu certo, mas usando a primeira atribuicao e não a segunda atribuicao.

Alguem pode explicar ?

... o c segue o padrão ascii e isto pode ocasionar erros quando se escreve com acentos voce. pode fazer o seguinte faça um programa que mostre os caracteres ascii em seu computador e veja quais o seu compilador mostra a programa deve seguir este algoritmo

Li a parte do curso relativa a modificadores de acesso de variáveis, na parte de variáveis "extern" e entendi qual a função delas, mas o que me deixou curioso foi como implementar programas que usem mais de um arquivo, o que não foi muito detalhado. Seria muito fora do escopo do curso pedir um exemplo de como fazer um programa que use mais de um arquivo, demonstrando como criá-lo e linká-lo em algum compilador usado no curso (por exemplo o RHIDE, que deve estar sendo usado pela maioria dos alunos), e aproveitando para demonstrar o uso de variáveis "extern" ??

Rodei o programa com float e deu certo. Agora fiquei com uma dúvida: e se a gente quiser utilizar variáveis inteiras tão grande, o que fazemos, choramos ? :-) Houvi falar em alguns emails de variáveis de ambientes. o que é isso e como se configura as mesmas?

Realmente o limite para long int é 2.147.483.647, e 10 elevado a 9 está dentro do limite, mas...10 elevado a 10 é maior que isso! Observe:

Limite máximo = 2.147.483.647

10 elevado a 10 = 10.000.000.000 (maior!)

Rodei o programa com float e deu certo. Agora fiquei com uma dúvida: e se a gente quiser utilizar variáveis inteiras tão grande, o que fazemos, choramos ? :-)

Pois eh, choramos. A unica saida eh "escovar bits" e usar arrays ou listas de inteiros para armazenar numeros muito grandes...

Professor Renato, socorra-nos!

Na Pergunta 5 da auto avaliação c390 "O trecho do programa a seguir eh"

```
#include <stdio.h>
main()
{
```

```

char condicao;    // declara
condicao = 'D';   // inicializa
int i = 1;        // declara e inicializa
printf("%c  %d",condicao,i);
}

```

Este trecho eh valido ou não, Se não poderiam me dizer o porquê???????

... porque no programa abaixo, que deve fornecer o preco de um aluguel de fita cassete em funcao do numero de dias e da cor do selo da fita(para cada cor um preco diario diferente), o programa na hora de fornecer o valor trava (este programa executou uma operacao ilegal)?

```

#include <stdio.h>
void main ()
{
char cor, verde, amarelo, branco, rosa;
float precototal;
float dias;
verde = 2,50;
amarelo = 2,70;
branco = 2,00;
rosa = 3,00;
printf("entre com o numero de dias\n");
scanf ("%f",&dias);
printf("Entre com a cor do selo\n");
scanf ("%s",&cor);
if (cor == verde)
{
    precototal = dias *2,50;
    printf("O valor a ser cobrado e.....RS %.2f\n",precototal);
}
if (cor == amarelo)
{
    precototal = dias*2,70;
    printf("O valor a ser cobrado e.....RS %.2f\n",precototal);
}
if (cor == branco)
{
    precototal = dias*2,00;
    printf("O valor a ser cobrado e.....RS %.2f\n",precototal);
}
if (cor == rosa){
    precototal = dias*3,00;
    printf("o valor a ser cobrado e..... RS %.2f\n",precototal);
}
}

```

Resposta: uso da virgula como separador decimal

... gostaria de saber como poderia fazer para acessar um elemento qualquer de minha string, ou seja quero deixar a disposicao do usuario, para que ele escolha qual elemento quer acessar tipo 1º letra, 2º letra ou qualquer outra, envio parte do programa, gostaria de ajuda , muito obrigado.

Voce esta cometendo um erro: primeiro voce declara «char cor, verde, amarelo, branco, rosa» e depois atribui a estas variaveis de tipo char valores float, e por ultimo ainda le com se fossem strings! olha so: uma string eh um array de chars, vc tem q declarar algo como «char[10] cor, verde, amarelo, branco, rosa» pra q essas variaveis sejam strings, ok?

Sobre sua outra duvida, as chaves sao para definir um bloco. isto eh nescessario pois existe mais de um comando a ser executado se o if for verdadeiro.

A expressao é: $z = y/x$;

y e x sao variáveis inteiras, portanto o resultado da divisao entre elas é um inteiro e nao um float. Sendo assim a variável z recebe um valor inteiro. Variáveis float podem receber valores inteiros (ela nao se TORNA um inteiro, apenas RECEBE este valor). Perceba que o contrário nao é verdadeiro: um inteiro nao pode receber um float (sem uma conversao).

Vejam o que a troca de um %c por um %s pode ocasionar...

Eu estava compilando e testando o exercicio "b) Escreva um programa que leia duas strings e as coloque na tela. Imprima tambem a segunda letra de cada string.", aula 2, "Introducao Basica as Entradas e Saidas".

Compilacao ok, cliquei em Run (compilador TCLITE).

O programa solicitou as duas strings, escreveu-as na tela e....travou tudo ! Nem CTRL+ALT+DEL resolveu, desliguei pelo botao, mesmo. Reiniciei o micro, abri o TCLITE, examinei o programa.... mandei executar novamente. Mensagem de erro recebida: "UNEXPECTED END OF RAM", ou algo parecido.

Tudo bem que meu micro eh um Lentium 100, mas se 48 de RAM nao fosse suficiente para executar o que o programa propunha, entao nem acessar essa lista eu conseguiria.

Reiniciei, examinei calmamente o programa, linha por linha.

Meu erro: usei %s ao inves de %c quando fui chamar os caracteres string1[1] e string2[1] conforme abaixo:

```
printf("\nPrimeira palavra digitada: %s",string1);
printf("\nSegunda palavra digitada: %s",string2);
/*LINHA ABAIXO EH A CORRETA*/
printf("\n\nAs segundas letras sao, respectivamente, %c e %c.",string1[1],string2[1]);
/*LINHA ERRADA*/
printf("\n\nAs segundas letras sao, respectivamente, %s e %s.",string1[1],string2[1]);
```

Entao, se alguem mais levar sustos como esse... a culpa nao eh da ram !

Dúvidas sobre operadores e expressões

Resumo:

- 1) como usar o operador aritmético mod
- 2) como usar o operador lógico de negação
- 3) como relacionar vários operadores numa expressão
- 4) uso da atribuição no lugar do operador de igualdade
- 5) uso de operadores unários e binários

Dúvidas Originais

Fiz um dos exemplos do curso e compilei o programa sem problemas, mas continuo tendo uma duvida. No programa abaixo:

```
if(!(i%2))
```

qual eh a ideia do comando (!(i%2)) contido no if? O que quer dizer isso?

.... exercicio : explique porque esta errado

```
if(num=10).....
```

nao entendi !

1 - na pagina de operadores aritmeticos e de atribuicao e dito : "O operador - como troca de sinal é um operador unário que não altera a variável sobre a qual é aplicado, pois ele retorna o valor da variável multiplicado por -1. ". NAO ENTENDI

Digamos que nós tenhamos duas variáveis: A=2 e B=3.

A instrucao: A=-B;

nao vai modificar o valor de B, como se poderia supor, mas apenas o de A

Resultado: A=-3 e B=3

2 - exercicio 11 da auto-avaliação online da aula 3

```
int a = 10, b = 20, c;
```

```
c = a+++b;
```

eu entendi assim: a e incrementado de 1 e passa a 11 ao ser somado com 20 da 31. porque a resposta 30 para c? da questao 12 entao realmente nao entendi nada. se alguem puder me ajudar

Vamos escrever a expressao de forma mais legível:

```
c = a++ +b
```

Lembra-se que o operador de incremento (++) quando colocado depois da variável só altera o seu valor no final da operação? Pois bem, vejamos como trabalha essa instrução.

```
c = a++ +b
```

O valor de b é somado ao de a e retornado a c Resultado parcial: a=10; b=20 e c = 30

E agora, finalmente o valor de a é incrementado pelo (++)

Resultado final: a=11; b=20 e c=30

Se a expressão fosse

```
c = ++a +b
```

então seu raciocínio estaria correto.

Aqui vc vê o quanto é necessário atenção ao programar em C. Um simples erro de lógica comprometeria o programa inteiro e seria muito difícil achar um erro tão pequeno num programa de algumas milhares de linhas.

3 - existe alguma função que resolva o problema do lixo que aparece quando se acentua as palavras nos programas para windows console que estamos estudando

Sinceramente eu não sei

No teste da aula 3, abaixo, fiquei em dúvida quanto a resposta do valor das variáveis

8- Diga o resultado das variáveis x, y e z depois da seguinte sequência de

operações:

```
int x,y,z;
```

```
x=y=10;
```

```
z=++x;
```

```
x=-x;
```

```
y++;
```

```
x=x+y-(z--);
```

Galera não entendi nada de unário e binário...

```
x=23;
```

```
y=x++;
```

```
x=24
```

```
y=23
```

Como isso acontece?

Dúvidas sobre funções

Resumo:

- 6) como usar a formatação do printf
- 7) uso de funções de biblioteca
- 8) compor um programa com mais de uma função
- 9) o que é o void
- 10) retorno de uma função
- 11) porque o scanf precisa do endereço da variável

Dúvidas Originais

Porque no programa abaixo, que deve fornecer o preço de um aluguel de fita cassete em funcao do numero de dias e da cor do selo da fita(para cada cor um preco diario diferente), o programa na hora de fornecer o valor trava (este programa executou uma operacao ilegal)?

```
printf("O valor a ser cobrado e.....RS %.2f\n", precototal);
```

O primeiro programa me saiu com o /n

por que em <stdio.h> tem esse .h

... não estou conseguindo desenvolver o programa de avaliação da pagina aula c240.html o 1 mais precisamente alguém poderia me dar uma luz to tentando assim mas não dá certo

```
#include <stdio.h>
int string ()
{
    char string[10];
    printf("\nDigite uma Letra: ");
    gets (string);
    printf ("\nVoce digitou: %s\n",string);
    return(0);
}
int main ()
{
    char Ch;
    Ch='D';
    printf ("%d",Ch);
    return(0);
}
```

}

Resposta: Poxa cara, q isso? vc pegou os dois primeiros programas da pagina, colocou um em baixo do outro e mudou o nome da main do primeiro pra string. q zona meu! po cara, eh auto-avaliacao, naum eh pra colar naum. :)

Eu fiquei travado aqui no primeiro exercício da aula 3 (c330). O que acontece eh que eu nao consigo fazer com que o printf mostre mais de uma variável. Ele só mostra o '10' das variáveis inteiras e o 'c' de coelha. Como eh que o printf mostra mais de uma variavel?

Acho que estou com um problema conceitual, devo estar fazendo alguma confusao, pois nao entendo quando se diz que a palavra VOID indica que a função nao retorna nada. Eu entendo que, no segundo exemplo da aula 2 a função retorna sim algo, que é o valor da divisão da variável Dias por 365,25 e no entanto este programa contém a palavra VOID. Por favor alguém pode me dizer onde está meu erro?

As funcoes retornam valores, em todos os exerci'cios que fizemos, mandamos a funcao "int main()" retornar um valor zero inserindo sempre o return (0) mas eu ainda naum entendi o porque se ao compilar todos os programas sem o return(0) não houve problemas, alguem poderia me explicar por favor?

Não entendi qual a função de return(0).

Alguem poderia me ajudar?

Li que e importante colocar & antes das variaveis quando se usa a função scanf() e mais tarde me deparei com uma variavel (dentro da funcao printf()) que nao usava o & e o programa funciona normalmente. Devo entender que & só deve ser usada nas variaveis que estao dentro da funcao scanf()?

Nos e-mails de duvidas vi em alguns programas a clausula VOID MAIN().

Qual e' a diferenca entre INT MAIN() e VOID MAIN()?

Dúvidas sobre comandos

Resumo:

- 1) quando é preciso colocar chaves nos comandos
- 2) como usar o while

Dúvidas Originais

Estreando na lista. Seguinte. eu tenho que fazer um trabalho pra amanhã, na qual a pessoa insere o valor em real e a contacao do dia do dolar, e ele converte para dolar. Eu estou com dois probleminhas. Um. eu nao entendi direito o funcionamento do while no C. Dois. Como formatar a saida do valor em dolar, tipo, eu soh quero colocar duas casas depois da virgula. Eu sei que eh um assunto jah meio avancado, mas eh que eh meio urgente... :0) Aqui vai o programinha ateh o ponto onde fiz. Gostaria de saber qual/quais as diferenças do float para o double pois em quase todas as linguagens de programação se usa as duas com o mesmo ponto de flutuação.

Porque no primeiro programa de exemplos sobre if codigo abaixo, é necessario abrir e fechar a funcao com chaves depois do segundo if, e nos demais nao?

```
#include <stdio.h>
```

```
int main(){
```

```
    int num;
    printf("Digite um numero: ");
    scanf("%d",&num);
    if(num>10)
        printf("\n\nO numero e maior que 10");
    if(num==10)
    {
        printf("\n\nVoce acertou!\n");
        printf("O numero e igual a 10.");
    }
    if(num<10)
        printf("\n\nO numero e menor que 10");
    return(0);
}
```

Identação (neologismo?): para o bem de todos nós, inclusive aqueles que já programam, sugiro um pequeno esforço para adquirir o hábito de idantar seus programas. Não dói nada tentar. É muito simples e rápido, bastar ter boa vontade no início. O que é idantar? É escrever um programa adotando um padrão bem definido com relação à coluna inicial de digitação de cada comando.

Para quê idantar? Para ficar claro. Para ficar fácil e rápido entender e diagnosticar problemas e dar manutenção num programa. Para errar menos ao se programar. Para ficar mais bonito. Fica fácil saber "o quê está dentro do quê", o quê está fora de um bloco de comandos, quando começa e termina um bloco de comandos.

Qual o padrão a se adotar na identificação? Há quem defenda um ou outro padrão. Desde que seja bom, e cumpra o seu objetivo, já será um avanço. O padrão que eu adoto pode ser visto acima: espaçamento de 2 espaços; se "subirmos a vista" na mesma vertical de um determinado }, o 1º caracter encontrado estará na linha que demarca o início de um bloco de comandos (um bloco de comandos é delimitado por um { no início e por um } no final); comandos pertencentes ao mesmo bloco de comandos se iniciam sempre na mesma coluna.

Dúvidas sobre ponteiros

Resumo:

- 1) passagem de parâmetros por referência

Dúvidas Originais

Quando fazemos um programa em C e utilizamos o comando "scanf ("%d", &z); "onde z é o nome da variável, se tirarmos o & da frente do z você digita um número qualquer e ele não pega o número que você digitou mas sim um número absurdo que não tem nada a ver com nada. Alguém sabe dizer porque isso acontece, uso o Turbo C++ versão 3.0 da Borland.

Dúvidas extras (Não tratadas pelo software educacional)

Resumo:

- 1) como configurar o compilador
- 2) como usar o compilador
- 3) como configurar o sistema operacional
- 4) como usar o teclado
- 5) como debugar o programa

Dúvidas Originais

Existe uma forma de configurar o ambiente, sem que tenhamos de alterar o AUTOEXEC.BAT.

Gostaria de saber se há algum bom compilador C/C++ em ambiente Windows 98 freeware e que dispense alteração do arquivo autoexec.bat. Os compiladores que constam na FAQ Page requerem alteração deste arquivo além de serem para ambiente DOS.

Pelo que parece neste curso o único que usa linux sou eu mas também sou relativamente novo nele o meu problema está sendo que não sei fazer funcionar bem o compilador aqui que é o xwpe e não estou podendo realizar os programas alguém tem alguma dica

Tenho todo um programa pronto em Turbo C++, ele funciona sem algum problema, a minha dúvida é a seguinte, como compilar este programa no Unix?

Alguem sabe me informar como faco as chaves dentro do programa? No meu copmutador nao existe mapa de caracteres para eu procurar..

Resposta: Os codigos ASCII das chaves sao:

{=<ALT>+123

}=<ALT>+125

Preciso fazer um trabalho para a faculdade e necessito dar o .exe com as opcoes de debug e outro .exe sem as opcoes de debug, no compilador que uso, o borland turbo c++ lite, nao encontro estas opcoes. Alguem poderia me dizer como faco para ativar estas opcoes no menu option, ja que no help elas aparecem, mas no editor nao...

Fiz os primeiros programas, mas ao ir em Compile/Make, não acontece nada. Em compile/compile o programa é compilado, mas depois, ao ir em Run aparece a seguinte mensagem:

Could not find the program. You must build at fists the program to use

this function, because the symbols ate taken from the file"

Será que o problema surge na hora do Save?

Bibliografia

- Baitello, N.J. (1998) O Ensino do jornalismo diante das novas tecnologias de comunicação. *Simpósio: Universidade e Novas Tecnologias – Impactos & Implicações*. Universidade de São Paulo. São Paulo.
- Binder, F. V. e Direne, A. I. (1999) Conceitos e Ferramentas para apoiar o ensino de lógica de programação imperativa. In: *Anais do X Simpósio Brasileiro de Informática na Educação*. Curitiba.
- Booch, G., Rumbaugh, J., Jacobson, I. (1999) *The Unified Modeling Language – User Guide* Addison-Wesley
- Brooks Jr., F.B. (1995) *The Mythical Man-Month* Addison-Wesley 20th anniversary Ed.
- Brusilovsky, P., Schwarz, E., Weber, G. (1996) ELM-ART: An Intelligent Tutoring System on World Wide Web In: C.Frasson, G.Gauthier e A.Lesgold(eds.): *Intelligent Tutoring Systems. Lecture Notes in Computer Science*. (pp.261-269). Berlin: Editora Springer
- Cortelazzo, I.B.C. e Oliveira, V. B. (1996) Repensando a Educação em função de mudanças sociais. In: *Oliveira, V. B.(org.): Informática em Psicopedagogia* (pp.111-130). São Paulo: Editora SENAC.
- Du Boulay, B. e O'Shea, T. (1981) Teaching novices programming In: *Coombs, M. J. e Alty, J. L. Computing Skills and the User Interface* (pp. 147-200). London: Academic Press Inc.
- English, S. e Yazdani, M. (1997) *Cooperative Learning in a Virtual University*. Department of Computer Science. University of Exeter. Exeter, United Kingdom
- Farrer, H., Becker, C.G. et al. (1999) *Pascal Estruturado*. 3^a Ed. Rio de Janeiro: Ed. LTC
- Flavell, J.H. (1975) *A psicologia do desenvolvimento de Jean Piaget*. Trad.: M.H.S.Patto. São Paulo: Ed. Pioneira.
- Galvis, A.H. (1997) Software Educativo Multimídia-Aspectos críticos no seu ciclo de vida. *Revista Brasileira de Informática na Educação*. No. 1, Set/97 Sociedade Brasileira de Computação.
- Heckel, P. (1993) *Software Amigável: Técnicas de projeto de software para uma melhor interface com o usuário* Trad.: Insight Serv. Inf. Rio de Janeiro: Ed. Campus.
- Hodges, M.E e Sasnett, R. M. (1993) *Multimedia Computing: case studies from MIT Project Athena* Addison-Wesley.
- Kernighan, B.W. e Ritchie, D.M. (1990) *C : A linguagem de programação padrão ANSI*. Rio de Janeiro: Ed. Campus.
- Kotonya, G e Sommerville, I (1998) *Requirements Engineering: process and techniques*. Inglaterra: Ed. Jonh Wiley & Sons.

- Lévy, P. (1993) *As Tecnologias da Inteligência*. Trad.: Costa, C. I. Rio de Janeiro: Editora 34.
- Litto, F. M. (1996) Repensando a Educação em função de mudanças sociais... In: Oliveira, V. B. (org.): *Informática em Psicopedagogia* (pp.85-110). São Paulo: Editora SENAC
- Lomônaco, J.F.B (1984-a) A natureza da aprendizagem. In Witter, G.P. e Lomônaco, J.F.B *Psicologia da Aprendizagem* -Temas básicos de Psicologia (pp1-7). São Paulo: Ed. Pedagógica e Universitária-EPU.
- Lomônaco, J.F.B (1984-b) Modelo cognitivo. In Witter, G.P. e Lomônaco, J.F.B *Psicologia da Aprendizagem* -Temas básicos de Psicologia (pp29-35). São Paulo: Ed. Pedagógica e Universitária-EPU
- Lomônaco, J.F.B (1984-c) Aprendizagem de conceitos. In Witter, G.P. e Lomônaco, J.F.B *Psicologia da Aprendizagem* -Temas básicos de Psicologia (pp59-71). São Paulo: Ed. Pedagógica e Universitária-EPU
- Martin, J. M. (1996) *Is Turing a better language for teaching programming than Pascal?* Dissertação (Mestrado) University of Stirling, Department of Computing Science. Escócia.
- Mattos, M.I.L. (1987) Aprendizagem e Informática. In Witter, G.P. e Lomônaco, J.F.B *Psicologia da Aprendizagem-Aplicações na Escola*-Temas básicos de Psicologia (pp56-71). São Paulo: Ed. Pedagógica e Universitária-EPU
- Oliveira, V. B. (1996) A compreensão de sistemas simbólicos. In Bossa, N. A. e Oliveira, V. B. *Avaliação psicopedagógica da criança de 7 a 11 anos* (pp15-45). Petrópolis: Ed. Vozes
- Oliveira, V. B. (1998) O Símbolo e o Brinquedo. 2a.Ed. Petrópolis: Ed. Vozes.
- Oliveira, V. B. (1998) Avaliação cognitiva do adolescente: uma proposta sintático-semântica. In: Bossa, N. A. e Oliveira, V. B. (org.): *Avaliação Psicopedagógica do Adolescente* (pp.13-55). Petrópolis: Editora Vozes.
- Oliveira, V. B. e Fischer, M. C. (1996) A Microinformática como instrumento da Construção Simbólica In: Oliveira, V. B. (org.): *Informática em Psicopedagogia* (pp.145-164). São Paulo: Editora SENAC.
- Pagano, R. L. (1992) *Computer Simulation as an Educational Tool*. Tese(Doutorado) 234p. Louvan-la-Neuve Université Catholique de Louvan
- Pressman,R.S. (1995) *Engenharia de Software* Trad: Santos,J.C.B. São Paulo: Editora Makron Books.
- Robertson, S. e Robertson, J. (1999) *Mastering the Requirements Process*. Inglaterra: Ed. Adison-Wesley.
- Rocha, H.V., Baranaukas, M.C.C. (2000) *Design e avaliação de interfaces humano-computador*. Escola de Computação 2000. Departamento de Ciência da Computação IME- USP. São Paulo

- Sack, W. (1992) Knowledge Base Compilation and the Language Design Game. In: *Intelligent Tutoring Systems, 2º Internacional Conference (Lectures Notes in Computer Science)* Eds: C. Frasson, G. Gauthier e G.I. Mccalla (p. 225-233). Berlin: Ed. Spring-Verlag.
- Self, J. (1995) *Computacional Mathetics: towards a Science of Learning System Designer*. University of Leeds. United Kingdom.
- Setzer, V. W. e Kon, F. (1995) *Introdução a rede Internet e seu uso* São Paulo: Ed. Edgard Blücher.
- Sommerville, I. (1995) *Software Engineering* 5ª. Ed. Inglaterra: Editora Addison-Wesley
- Stokes, D.A. (1993) Requirements Analysis In: *Mc Dermid, J.A. (ed): Software Engineer's Reference Book*. Capítulo 16. Florida: CRC Press Inc.
- Valente, J. A. (1998) Diferentes usos do Computador na educação. In: *J. A. Valente (org.), Computadores e Conhecimento - Repensando a educação* (pp. 1-27) 2ª. Ed. Campinas: UNICAMP/NIED.
- VanLehn, K. (1996) Conceptual and Meta Learning during coached problem solving In: C. Frasson, G. Gauthier e A. Lesgold (eds.): *Intelligent Tutoring Systems. Lecture Notes in Computer Science*. (pp. 29-47). Berlin: Editora Springer.
- Viccari, R.M., Giraffa, L.M.M. (1996) Sistemas Tutores Inteligentes: abordagem tradicional X abordagem de agentes In: *XIII Simpósio Brasileiro de Inteligência Artificial. Sociedade Brasileira de Computação Tutorial*. Curitiba
- Wasselius, Jacco e Ververs, Frans. (1990) *Some elementary questions on software quality control* Software Engineering Journal, Nov 1990 (p. 319-330)
- Wirth, N (1985) *Programação sistemática em Pascal*. 4ª Ed.. Rio de Janeiro: Ed. Campus.
- Witter, G.P (1984) Modelos behavioristas. In Witter, G.P. e Lomônaco, J.F.B *Psicologia da Aprendizagem* -Temas básicos de Psicologia (p. 9-28). São Paulo: Ed. Pedagógica e Universitária-EPU
- Yazdani, M (1986) *Intelligent Tutoring Systems: An overview* . *Expert Systems*, vol 3 (p 154-162). Department of Computer Science, University of Exeter, Exeter, United Kingdom.

Referências de Endereços Eletrônicos na Internet

Andes: <http://www.pitt.edu/~gertner/andesins.html>

Aplication Program Tutor – APT: <http://www.dpsnet.com/apt/index.html>

Curso de Linguagem C – UFMG: <http://ead1.eee.ufmg.br>

ELM- ART: <http://www.contrib.andrew.cmv.edu/~plb/home.html>

Projeto Mac Multimídia: <http://www.ime.usp.br/~macmulti>