Desenvolvendo o ORKUT em C – uma nova abordagem para ensino de linguagem de programação estruturada

Frederico Brito Fernandes[1], Micheline Carvalho Barroso[1], Marcelo Leal Spinelli[1], Cláudia Batista Melo[2], Alisson Vasconcelos de Brito[2]

[1]Instituto de Educação Superior da Paraíba (IESP)
Avenida João Maurício, 1801 – Bessa – João Pessoa – PB. CEP: 58037-010
CGC: 70.118.716 / 0001-73 Telefone: (83) 214–9200

[2]Coordenação de Pós-Graduação em Engenharia Elétrica – Universidade Fereral de Campina Grande (UFCG) – Campina Grande – PB

fredbf@gmail.com, micheline.barroso@gmail.com, mlspinelli@gmail.com, claudiamelo@dee.ufcg.edu.br, alisson.brito@gmail.com

Abstract. *This paper aims to define a methodology based on a top-down approach for teaching imperative programming languages. The greater difficult in learning a new programming language is the lack of understanding of learning various concepts without a practical example, commercial or not. Our approach suggests the developing of an application during the class, since from requirements analysis, data modeling, and the development itself, where all programming concepts are explored. We intend to improve students' abilities like team development, project management, modeling and analysis, to show them how important are these knowledge in their professional life.*

Resumo. *Este trabalho visa definir uma metodologia que utiliza uma abordagem top-down para ensino de linguagem de programação estruturada. A maior dificuldade dos alunos em aprender uma linguagem de programação é a desmotivação em aprender todos os conceitos sem visualizar uma aplicação prática, comercial ou não. Nossa abordagem sugere o desenvolvimento em sala de aula de uma aplicação, desde a parte de análise de requisitos, modelagem dos dados, até a implementação, onde serão explorados todos os conceitos de programação. Vislumbramos também desenvolver nos alunos a capacidade de trabalhar em equipe, conceitos de gerência de projeto, modelagem e análises para que eles possam perceber a importância dessa formação em suas vidas profissionais.*

Palavras-chaves: metodologia de ensino top-down, linguagem de programação, desenvolvimento de aplicações

1. Introdução

Nenhum conceito deve ser ensinado em uma disciplina de linguagem de programação. Essa frase, que parece exagerada a princípio, acaba influenciando grande número de adeptos entre professores que se adequam à nova realidade de competências e habilidades proposta pelo MEC [10][11].

Não temos intuito de causar polêmica, nem entrar no mérito da questão. Porém, a grande totalidade de professores que ensinam linguagens de programação afirma que "*programar se aprende em casa*", e poucos sabem responder de imediato, quando questionados, qual a razão do grande índice de repetência e da falta de motivação dos alunos. Aliás, a resposta é quase sempre a mesma: "*na minha época eu estudava*". Esta

resposta baseia-se na experiência pessoal de que, quando éramos estudantes, participamos de grandes projetos, incentivados por bolsas do CNPq e PIBIC, onde tivemos a oportunidade de complementar os conhecimentos adquiridos em sala de aula.

Novas metodologias e idéias sobre abordagens de ensino de programação e de algoritmos surgem [1][4][5][7][8]; os currículos estão sendo redefinidos de forma mais flexível, aberta e interativa; as competências e habilidades tomam lugar de destaque na educação brasileira, também na área de computação e informática [10][11]; assim como, o desenvolvimento de projetos práticos e interdisciplinares e o uso de múltiplos conhecimentos, por ora chamados de competências, motivam e melhoram o desempenho dos alunos [1].

Este trabalho apresenta uma nova abordagem de ensino de linguagem de programação estruturada, cuja idéia principal baseia-se no desenvolvimento em sala de aula de uma aplicação, desde a parte de análise de requisitos, modelagem dos dados, até a implementação, onde serão explorados todos os conceitos da linguagem estudada.

A seção seguinte (seção 2) descreve as abordagens tradicionais de ensino de linguagens de programação. Nossa proposta de metodologia foi descrita na seção 3. A seção 4 define o produto a ser desenvolvido. Finalizando, a seção 5 apresenta as considerações finais acerca da metodologia proposta.

2. Metodologias Tradicionais

Disciplinas relacionadas ao ensino de linguagens de programação representam parte substancial da estrutura curricular da maioria dos cursos superiores na área de informática. O raciocínio lógico, adquirido e exercitado ao longo destas disciplinas, contribui, de forma decisiva, para o pleno aprendizado dos demais conceitos abordados durante o curso, não necessariamente aqueles diretamente associados à programação. Podemos afirmar que o raciocínio lógico, explorado durante o desenvolvimento de programas de computador, representa a essência da computação, e, conseqüentemente, é indispensável à formação dos futuros profissionais da área.

Segundo Nicholson [8], o objetivo inicial de qualquer curso da área de computação é formar profissionais com diversas habilidades na área de programação: a habilidade em programar utilizando várias linguagens de programação, e a habilidade em aprender novas linguagens de programação. Dessa forma, ao estudar uma linguagem de programação, o aluno deve ser apresentado a determinados conceitos que, além de garantir o aprendizado da linguagem em questão, servirão de base para o aprendizado de novas linguagens quando necessário.

Linguagens de programação são, geralmente, ensinadas em disciplinas independentes, totalmente dedicadas ao conhecimento de uma linguagem específica e com pouca, ou quase nenhuma relação com os demais requisitos necessários ao desenvolvimento de sistemas com qualidade (Metodologias para Análise de Sistemas; Modelagem, Projeto e Implementação de Banco de Dados; entre outros). Ao longo das aulas ministradas, os alunos são apresentados às estruturas sintática e semântica da linguagem, ilustradas através de exemplos de trechos de código e exercitadas em listas de exercícios desenvolvidas em laboratório. Estas listas são compostas por exercícios simples e genéricos, direcionados à prática dos conceitos discutidos no momento. O conhecimento adquirido é avaliado através de provas periódicas (escritas e/ou práticas), da resolução de exercícios aplicados durante as aulas e, alternativamente, do desenvolvimento de algum projeto como produto final da disciplina.

Uma das maiores limitações deste tipo de abordagem, de acordo com Nicholson [8], é que os alunos não são direcionados a trabalhar em exemplos de código mais complexos e sofisticados. Como conseqüência, eles produzem códigos fracamente estruturados, que, provavelmente, empregam recursos inadequados da linguagem, tornando-os incapazes de explorar o conjunto completo de recursos necessários ao desenvolvimento de sistemas. Além disso, a incapacidade de vislumbrar ambientes de programação de sistemas reais culmina em um baixo nível de motivação para o aprendizado de programação.

Por fim, observa-se que são nessas disciplinas, ligadas diretamente ao segmento de programação, onde pode ser encontrado um maior número de alunos despreparados. Este fato pode ser comprovado pelo alto grau de repetência nas disciplinas introdutórias, bem como pelas dificuldades demonstradas pelos alunos nas disciplinas mais avançadas que exigem conhecimento prévio em programação [9]. Dessa forma, observa-se que as metodologias tradicionais de ensino de linguagens de programação contribuem para criar barreiras no aprendizado da programação e, o mais agravante, propagar este problema, gerando um fluxo contínuo de alunos, com grau de interesse cada vez menor em programação, ao longo do curso.

3. Metodologia Proposta para Ensino de Linguagem de Programação Estruturada

Nossa proposta é de ensinar uma linguagem de programação estruturada a partir do desenvolvimento de um projeto em sala de aula, do começo ao fim. Na metodologia tradicional, o professor ensina conceitos da linguagem nova, e depois os alunos praticam com exercícios de programação super didáticos, porém com excesso de matemática, trigonometria e álgebra. A idéia é mostrar como se desenvolve a solução de um problema real, e ensinar os conceitos em paralelo. Acreditamos que essa abordagem top-down seja motivadora e o processo de ensino-aprendizado seja melhor aproveitado na disciplina.

Fazendo uma analogia com o seriado norte-americano, entitulado *24 horas*, que conta um trama policial a cada temporada, de 6 meses de duração. Cada episódio é exibido nas quintas-feiras, com 1 hora de duração, contando um pedacinho da trama. Cada episódio é fundamental para o entendimento da temporada. Agora perceba que o telespectador pode deixar de assistir 25% dos episódios por lei. Em outros 25% o telespectador estava com problemas de dinheiro, familiares e profissionais, e não prestou a devida atenção. Imagine ainda que contratempos aconteçam, como falta de energia, interferência no sinal do satélite devido à chuva, entre outros. Enfim, o último capítulo, em vez de concluir todo o processo desencadeado na temporada, poderá ser muito traumático para o telespectador, por achar que o diretor poderia ter feito melhor. Perceba que, se o telespectador pudesse ver o último capítulo no início da trama, apesar de não entender muita coisa, poderia ajudá-lo a reconstruir intuitivamente os episódios perdidos, pois ele sabe a direção a ser seguida. Temos intenção de não subestimar a capacidade mental dos alunos e de motivá-los a caminhar em seus ritmos.

Na maioria das matrizes curriculares dos cursos de computação e informática, as disciplinas de programação estruturada aparecem nos dois primeiros períodos. Apesar dos alunos, em geral, não terem cursado disciplinas como Análise e Projeto de Sistemas, Engenharia de Software e Bancos de Dados, achamos interessante que ele visualize a necessidade desse conhecimento a posteriori.

Conceitos básicos de análise de requisitos e modelagem dos dados não devem ser cobrados como objeto de avaliação no projeto. Todo o processo de análise já deve estar formalizado para que seja compreendido em poucas aulas. Entretanto, fazer um paralelo na modelagem de dados torna-se adequado, mostrando ao aluno que, uma vez definidos todos os requisitos, eles podem ser mapeados em funções em uma linguagem estruturada ou classes em uma linguagem orientada a objetos. Novamente, uma vez definido um modelo conceitual, como o Diagrama de Entidade-Relacionamento, ele poderá implementá-lo usando árvores e listas encadeadas em arquivos texto, tabelas relacionais, ou mesmo objetos geográficos.

3.1 Domínio

Segundo Flatt [12], a programação deveria ser ensinada como extensão do que os estudantes já conhecem. A escolha do domínio da ferramenta a ser implementada é crucial para validação de nossa metodologia, pois a maioria dos requisitos deve ser conhecida pelos alunos. O principal objetivo aqui é de explorar o reaproveitamento de código para a resolução de problemas práticos. Consideramos que esse é um tópico muito importante, pois a Internet e o avanço de soluções integradas de componentes requerem um profissional de informática capaz de integrar vários módulos.

A *extensibilidade* da ferramenta é outro fator determinante nessa escolha. Uma vez que admitimos que os alunos possuem diferentes níveis e velocidades de raciocínio lógico, é imprescindível que a ferramenta possa ser estendida, a partir da criação e implementação de novos requisitos. Por questões comerciais, uma ferramenta simplesmente didática não deve ser adotada, como implementar um jogo de cartas, ou qualquer outro jogo qualquer. É interessante mostrar ao aluno que ele poderá adaptar a solução proposta para um domínio específico que realmente solucione um problema real.

3.2 Dinâmica da Aula

Sugerimos que o quadro branco seja dividido em duas partes: (Metade 1) onde o professor irá desenvolver a implementação e (Metade 2) onde serão ensinados os conceitos a serem abordados naquela implementação. Ideal se aula fosse ministrada com auxílio de Data Show, pois enquanto o professor implementasse os requisitos, dicas e macetes do compilador utilizado fossem também ensinados aos alunos. O quadro branco poderia ser utilizado para explicar os conceitos da linguagem com algum pequeno exemplo didático. Ainda se a faculdade dispuser de recursos, algum programa para gravação de cada aula, como o Camtasia Studio[1], poderia ser usado para gravar vídeo e áudio do professor implementando os requisitos.

3.3 Sistemática de Ensino do Conteúdo Programático

Toda disciplina possui um objetivo a ser alcançado. Para uma disciplina de linguagem de programação estruturada podemos citar os macro conceitos: Função, arquivo, estrutura, ponteiro, alocação dinâmica, funções de Entrada e Saída (E/S), vetores, strings, macros, operadores relacionais e lógicos, entre outros. Torna-se necessário dizer que os conceitos devem ser ensinados gradativamente e de forma acumulativa. Isso permitirá, por exemplo, o uso exaustivo de funções pelos alunos, pois acreditamos que esse tópico deveria ser ensinado no início de qualquer curso de

[1] http://www.techsmith.com/products/studio/default.asp

programação, uma vez que, um programa estruturado pode ser visto como uma função que chama outras.

Não pretendemos estabelecer uma forma única de sistemática de ensino, porém, os conceitos de uma linguagem de programação estruturada poderiam ser ensinados da seguinte forma:

a) Interface – função, funções de E/S, operadores relacionais e lógicos, estruturas de controle e repetição. Aqui o aluno poderá criar a interface inicial do sistema, praticando a sintaxe básica dos principais comandos da linguagem.

b) Estruturação dos Dados – tipo, estrutura, string e vetores. Nessa etapa, o aluno poderá aprender sobre os tipos de dados, definir tipos novos usando *estrutura* para o usuário e todas as outras entidades, preencher vetores com todos os usuários, usar funções de string para buscar, comparar usuários no vetor.

c) Armazenamento dos Dados – arquivos, ponteiro e alocação dinâmica de memória. Nesse ponto o aluno irá implementar o mecanismo de persistência dos dados. Conceitos mais avançados da linguagem, como ponteiros e alocação dinâmica de memória devem ser explorados.

d) Implementação das Funcionalidades – resto dos conceitos.

Vale ainda insistir que, exceto na Interface, os requisitos são implementados em todas as partes (b, c e d), de forma gradativa e acumulativa. Quando o professor estiver ensinando na seção *Armazenamento de Dados*, não implica dizer que ele unicamente está preso a tipo, estrutura, string e vetor. Mesmo porque, o conceito de *tipo* foi brevemente visto na Interface, porém não explorado como deveria ter sido feito. Essa divisão serve apenas como orientação para definir a seqüência do conteúdo a ser avaliado pelo professor.

3.4 Avaliações

Sugerimos que, em cada nota, a implementação dos requisitos tenha peso 3 e a prova teórica com peso 7. A prova teórica poderia ter questões do tipo "codifique o requisito X do projeto acrescentando a característica Y". Dessa forma, o aluno pode ser avaliado de forma prática, simples, e o professor não precisa se preocupar tanto com o problema da cópia das implementações.

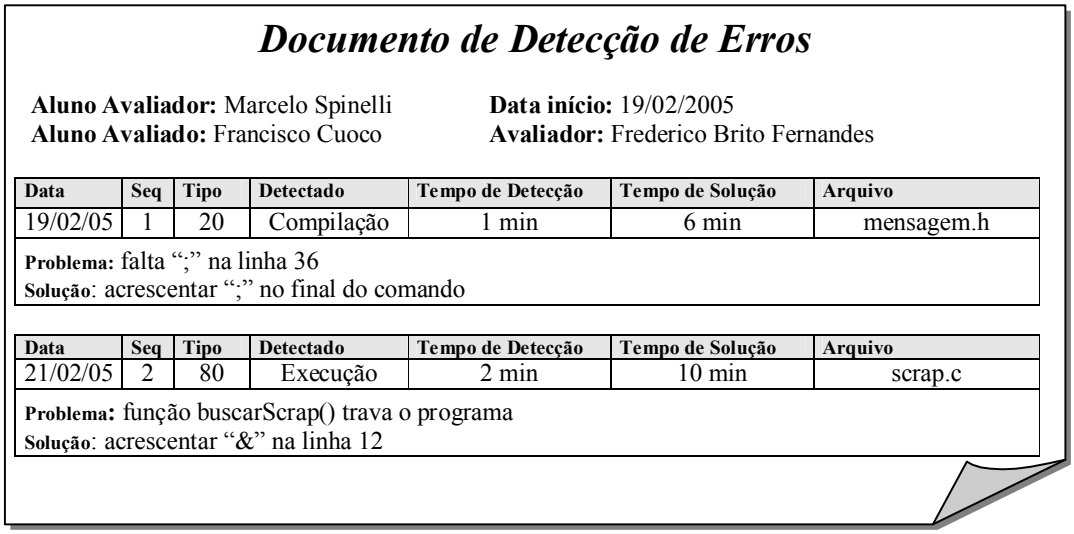
3.4.1 Do Projeto

Avaliação Individual

No decorrer da disciplina, o professor irá corrigindo as implementações dos requisitos, que serão avaliados de acordo com o mapeamento dos requisitos para funções definidos no projeto (ver Tabela 5).

Avaliação Coletiva

Bônus poderão ser alcançados na avaliação do trabalho dos colegas. Pretendemos trabalhar aqui a necessidade de estabelecer critérios na comunicação da falha detectada e na possível solução. Utilizaremos um documento de detecção de erros, inspirados nesse mesmo documento definido no Processo de Software para Times, proposto por Humphrey [6]. O professor será responsável por repassar esse documento ao aluno avaliado.



Documento de Detecção de Erros

Aluno Avaliador: Marcelo Spinelli **Data início:** 19/02/2005
Aluno Avaliado: Francisco Cuoco **Avaliador:** Frederico Brito Fernandes

Data	Seq	Tipo	Detectado	Tempo de Detecção	Tempo de Solução	Arquivo
19/02/05	1	20	Compilação	1 min	6 min	mensagem.h
Problema: falta ";" na linha 36 **Solução**: acrescentar ";" no final do comando						

Data	Seq	Tipo	Detectado	Tempo de Detecção	Tempo de Solução	Arquivo
21/02/05	2	80	Execução	2 min	10 min	scrap.c
Problema: função buscarScrap() trava o programa **Solução**: acrescentar "&" na linha 12						

Figura 1 - Documento de detecção de erros

Código	Tipo	Explicação
10	Documentação	Comentários, mensagens
20	Sintaxe	Erros nos comandos da linguagem, erros de datilografia
30	Empacotamento	Gerenciamento de mudanças, bibliotecas e versões
40	Atribuição	Variáveis: declaração, nomes duplicados, escopos, limites
50	Interface	Atende aos requisitos estabelecidos
60	Checagem	Mensagens de erro, checagem inadequada
70	Dados	Estrutura, conteúdo dos dados, arquivos
80	Função	Lógica: se a função realmente faz o que deveria fazer, parâmetros
90	Sistema	Distribuição dos sistema em arquivos
100	Ambiente	Configuração do compilador

Tabela 1 - Tabela de especificação de erros

3.4.2 Do Relatório Técnico

Ao final da disciplina, os alunos deverão escrever um relatório técnico descrevendo sua análise crítica da metodologia. Nesse relatório, o aluno irá definir todas as etapas do projeto, dificuldades encontradas, resultados alcançados, soluções propostas, análise crítica de todas as etapas da metodologia na disciplina ressaltando 3 vantagens e 3 desvantagens.

4. Estudo de Caso: ORKUT

Optamos em desenvolver o ORKUT (www.orkut.com), uma ferramenta de comunicação social amplamente difundida entre os jovens. Dois foram os motivos principais nessa escolha: (1) a análise de requisitos é de fácil compreensão e muito conhecida pelos alunos e (2) toda empresa gostaria de uma ferramenta de comunicação entre seus funcionários.

Admitindo ainda que alguns alunos terminarão a implementação dos requisitos antes de outros, torna-se interessante apoiá-los a desenvolver, em horário extra classe, uma adaptação dos requisitos já implementados para uma realidade específica. Imagine o ORKUT sendo a ferramenta possivelmente adaptada e usada na realidade acadêmica, por exemplo, onde seriam acrescentadas funcionalidades como "Estante de Arquivos",

onde os alunos colocariam arquivos dos projetos na área dos professores para futura correção, “Listas de Exercícios”, etc. Mudando o foco, imagine a realidade de uma empresa de desenvolvimento, acrescentando requisitos como “Criação de Projetos”, “Distribuição dos Módulos”, “Calendário de Entrega dos Módulos”, “Porcentagem dos módulos implementados individualmente e por equipe de trabalho”. Enfim, todas essas aplicações necessitam de interação entre pessoas e equipes.

4.1 Análise de Requisitos

O ORKUT é uma ferramenta de comunicação social amplamente difundida na WEB. Não é nosso interesse descrever todos os requisitos, porém podemos citar:

(1) Amigos – quando um usuário do ORKUT se autentica no site, pode-se ver a foto de todos os amigos dele. Na realidade, o usuário só pode se cadastrar no site, por convite de outro. Assim que se cadastra, ele já possui um único amigo. Porém, ele começará a construir sua rede de amizades;

(2) Comunidades – criadas para reunir um grupo de amigos com um mesmo objetivo. Assemelha-se com um fórum de discussão;

(3) ScrapBook – mural de recados que cada usuário possui. A diferença é que a mensagem fica ao lado da foto do usuário que deixou o recado (scrap). Todos podem olhar os scraps de todos. Assim que um usuário insere um scrap pra um amigo, esse scrap já fica disponível no ScrapBook do amigo;

(4) Mensagem – mecanismo de comunicação sigiloso. Ao contrário do scrap, o usuário pode ler unicamente suas próprias mensagens;

(5) Testemunho – o que os amigos dizem do usuário. Uma vez que o usuário escreve um testemunho, ele precisará da aprovação do amigo que recebeu para ser publicado.

Iremos descrever na Tabela 2 abaixo, todos os requisitos funcionais que iremos usar em nossa metodologia.

Conceito	Requisito	Descrição
1. Amigo	1.1. Inserir	Adiciona um amigo ao usuário
	1.2. Apagar	Exclui um amigo do usuário
	1.3. Buscar	Busca um amigo através do código do amigo

Tabela 2 - Requisitos funcionais do ORKUT

4.2 Modelagem dos Dados

Utilizaremos a tradicional Modelagem Entidade-Relacionamento para definirmos nosso modelo conceitual.

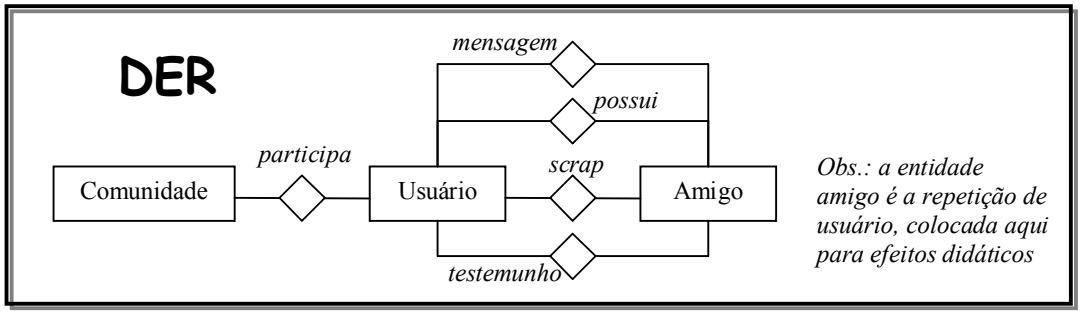


Figura 2 - Diagrama de Entidade-Relacionamento do ORKUT para algumas funcionalidades

Nosso DER acima poderia ser facilmente mapeado para tabelas em um SGBD relacional, mas como estamos querendo trabalhar conceitos de programação, iremos fazer a transformação para arquivos.

O primeiro passo é transformarmos as entidades *Comunidade* e *Usuário* nos arquivos comunidade.txt e usuário.txt (ver Tabela 3). Perceba que a data foi desmembrada em 3 inteiros, sendo necessários construir funções para manipular essa informação.

usuario.txt[2]							
codigo	nome	Sobrenome	telefone	dataNasc			email
				dia	mes	ano	
Int	char [10]	char [20]	char [20]	int	int	int	char [30]

Tabela 3 - Entidades mapeadas para arquivos

Em seguida, transformamos os relacionamentos Scrap, Testemunho, Participa e Possui para os arquivos scrap.txt, testemunho.txt, participantes.txt e amigos.txt, respectivamente.

scrap.txt, testemunho.txt e mensagem.txt					
codigo	codUsuario	dataCriacao			texto
		dia	mes	ano	
int	Int	byte	byte	byte	char [200]

Tabela 4 - Relacionamentos mapeados para arquivos

Porém, perceba que os arquivos *comunidades.txt* e *participantes.txt* devem ser únicos e compartilhados por toda a aplicação. Portanto, eles devem ser armazenados na pasta principal da aplicação (Figura 5).

4.3 Definição das Estruturas de Dados

Apesar de trabalharmos no paradigma estruturado, podemos definir cada entidade e relacionamento do DER (Figura 2) como uma estrutura, a fim de implementarmos os requisitos baseados nessas estruturas.

Data	typedef struct { int dia; int mes; int ano; } tData;

Comunidade	typedef struct { int codigo; char [30] nome; int codCriador; tData dataCriacao; char [200] descricao; } tComunidade;

Figura 3 – Estruturas em C que definem cada requisito

4.4 Mapeamento dos Requisitos para Protótipos de Funções em C

Todos os requisitos serão mapeados para protótipos de funções, conhecidos também como cabeçalhos das funções. Por exemplo, iremos definir todos os cabeçalhos relativos ao requisito *Scrap* no arquivo scrap.h. Esse arquivo ainda armazenará todas as variáveis, vetores e estruturas (por exemplo, o tipo *tScrap* descrito na Figura 3) relativas ao scrap. Perceba que o arquivo scrap.c deverá implementar todas essas funções e, evidentemente, importar o arquivo scrap.h.

[2] Perceba que usuário.txt teria as mesmas informações de amigos.txt, sendo desnecessário criar esse arquivo

AMIGO.H		
Func.	**Função em C**	**Retorno**
1.1	int inserirAmigo(int codUsuario, int codAmigo)	*0 para sucesso e 1 para código inexistente*
1.2	int apagarAmigo(int codAmigo)	*0 para sucesso e 1 para usuário inexistente*
1.3	tAmigo buscarAmigo(int codAmigo)	*o amigo e NULL caso codAmigo não exista*

Tabela 5 - Protótipos das Funções que definem os requisitos

4.5 Projeto de Interface

Como disciplina de programação estruturada, temos o objetivo de fazer com que o aluno desenvolva em C, uma interface simples, apenas para utilizar os requisitos implementados. Abaixo estão os front-end para nosso ORKUT.

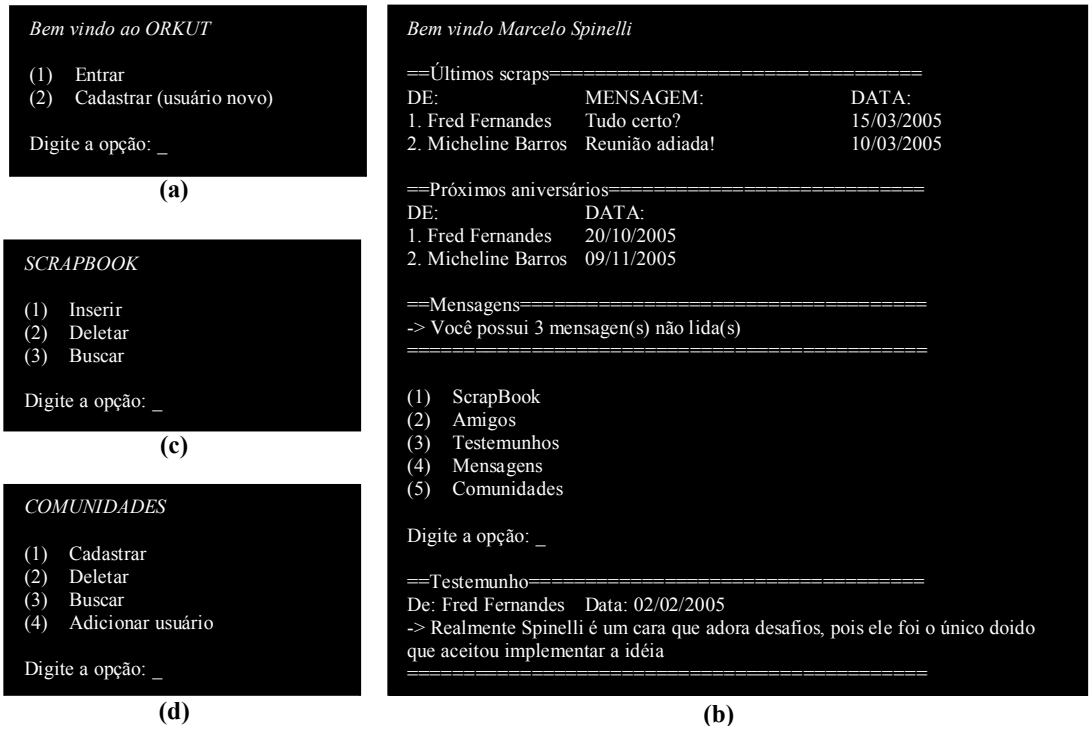


Figura 4 - Interface desejável do ORKUT em C. Tela de entrada onde o usuário pode se autenticar ou se cadastrar (a). Tela de entrada (b). Tela do ScrapBook (c). Tela das Comunidades (d)

Entretanto, como a idéia dessa metodologia é também de contemplar o desenvolvimento de uma ferramenta comercial. Para tanto, disponibilizamos uma interface desenvolvida em CGI (Common Gateway Interface) que permitirá a utilização de todas as funcionalidades implementadas pelos alunos para publicação na Web.

4.6 Amazenamento do Projeto

Toda a aplicação deverá ficar disponível em uma pasta compartilhada por todos os alunos. O aluno poderá ver sua aplicação comparada com a aplicação dos demais.

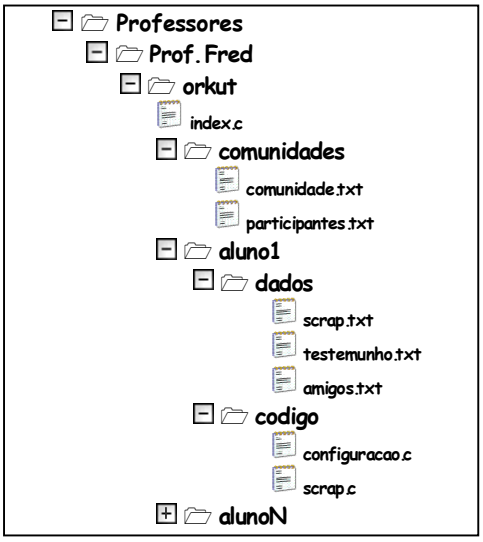


Figura 5 – Pastas para armazenamento da aplicação

5. Conclusão

Este trabalho sugere a utilização de uma abordagem top-down para o ensino de linguagens de programação estruturada. A metodologia de ensino proposta procura estimular o aluno a entender todo o processo de desenvolvimento de um sistema, do começo ao fim, dando um pré-embasamento teórico sobre conceitos que ele ainda irá adquirir, como em: (1) Banco de Dados, entendendo a modelagem de dados MER; (2) Análise de Requisitos, explanando sobre a criação de um documento que irá gerar uma único resultado; (3) Engenharia de Software, explicando documentos básicos dos processos de desenvolvimento PSP e TSP, assim como ferramentas para colaboração de código à distancia; (4) Competitividade e Inteligência, onde os alunos implementam os requisitos que estão faltando no aplicativo e constroem a interface da melhor forma possível, simulando realmente o trabalho em uma empresa de desenvolvimento.

Apesar de nos concentrarmos, praticamente 90% do tempo em sala no ensino da linguagem imperativa, que realmente é o objetivo central da metodologia, o aluno terá a oportunidade de aprender a estruturar de forma elegante e concisa todo o processo de desenvolvimento do objeto de estudo.

A partir do uso da nossa metodologia, podemos caracterizar algumas contribuições e fatores interessantes: (1) *vendabilidade* do projeto, pois o objeto de estudo da disciplina deve ser bem escolhido, a fim de poder ser estendido pra algum sub-domínio específico; (2) motivação dos alunos, pois percebemos que a metodologia, e principalmente o objeto a ser desenvolvido, o ORKUT, proporcionou uma motivação especial nos alunos, levando a alguns implementarem alguns requisitos comentados na primeira aula; (3) resgate de alunos desmotivados (4) forçar o aluno a pensar antes de programar; (5) visão geral do desenvolvimento de software, pois permite ao aluno aprender princípios básicos de análise e modelagem de dados mesmo sem ter cursado essas disciplinas; (6) ensinar a importância de codificar bem, pois outros alunos irão avaliar o projeto e (7) elaboração de relatório técnico.

Atualmente, a metodologia proposta está sendo aplicada em turmas do Curso de Bacharelado em Sistemas de Informação do IESP e do Curso Superior de Desenvolvimento de Sistemas da FATEC-PB. Como trabalho futuro, iremos propor uma extensão da nossa metodologia, a fim de contemplar outros paradigmas de programação.

6. Referências

[1] ALLAN, V. H.; KOLESAR, M. V., (1996). Teaching computer science: A problem-solving approach that works. *Proceedings of the Annual National Educational Computing Conference*. Minneapolis, MN, 1996.

[2] BEAUBOUEF, T.; LUCAS, R. and HOWATT, J. The UNLOCK System: Enhancing Problem Solving Skills in CS-1 Students. *SIGCSE Bulletin—inroads*., 2001.

[3] CORTE, Camila K. D.; BARBOSA, Ellen F. e MALDONADO, José C. Ensino Integrado de Fundamentos de Programação e de Teste de Software. *Anais do XXIV Congresso da Sociedade Brasileira de Computação*. Salvador, Bahia, 2004.

[4] DELGADO, Carla; XEXEO, José A. M.; SOUZA, Izabel F.; CAMPOS, Márcio e RAPKIEWICZ, Clevi E. Uma Abordagem Pedagógica para a Iniciação ao Estudo de Algoritmos. *Anais do XXIV Congresso da Sociedade Brasileira de Computação*. Salvador, Bahia, 2004.

[5] HAYNES, Christopher T. Experience with an analytic approach to teaching programming languages. 350-354. *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education*. Atlanta, Georgia, 1998.

[6] HUMPHREY, Watts S. Disciplined Software Teams. Nos anais da IX CITS Conferência Internacional de Tecnologia de Software: Qualidade de Software, Curitiba-PR, pp. 3-31, 1998.

[7] KOLIVER, Cristian; DORNELES, Ricardo V.; CASA, Marcos E. Das (Muitas) Dúvidas e (Poucas) Certezas do Ensino de Algoritmos. *Anais do XXIV Congresso da Sociedade Brasileira de Computação*. Salvador, Bahia, 2004.

[8] NICHOLSON, A. E. and FRASER, K. M. Methodologies for teaching new programming languages: A case study teaching LISP. *Second Australasian Computer Science Education Conference*. Melbourne, Australia, 1997.

[9] OMAR, N.; FRANÇA, Vilma F.; PIMENTEL, Edson P. A Caminho de um Ambiente de Avaliação e Acompanhamento Contínuo da Aprendizagem em Programação de Computadores. *III Workshop de Informática na Educação Computação do Estado de Minas Gerais*, 2003, Poços de Caldas - MG. III Workshop de Informática na Educação Computação do Estado de Minas Gerais, 2003. v. 1.

[10] REZENDE, Luiziana; SEGRE, Lídia M. e CAMPOS, Gilda H. B. O Modelo de Competências e as Implicações para o Currículo do Curso de Ciência da Computação. *Anais do XXIV Congresso da Sociedade Brasileira de Computação*. Salvador, Bahia, 2004.

[11] SOUZA, Alexandre P.; NETO, Wilson C. B. O Processo de Criação do Curso de Sistemas de Informação da Uniplac através da definição de competências. *Anais do XXIV Congresso da Sociedade Brasileira de Computação*. Salvador, Bahia, 2004.

[12] VAN ROY, Peter; ARMSTRONG, Joe; FLATT, Matthew and MAGNUSSON, Boris. The role of language paradigms in teaching programming. *ACM SIGCSE Bulletin*. Publisher ACM Press. New York, NY, USA, 2003.