UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS PATO BRANCO
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

SIMONE BAMPI

SISTEMA DESKTOP PARA REALIZAÇÃO DE TESTES DE MÚLTIPLA ESCOLHA

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2013

SIMONE BAMPI

SISTEMA DESKTOP PARA REALIZAÇÃO DE TESTES DE MÚLTIPLA ESCOLHA

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientadora: Profa. Beatriz T. Borsoi

PATO BRANCO
2013

ATA Nº: **209**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DA ALUNA **SIMONE BAMPI**.

Às 16:00 hrs do dia 18 de abril de 2013, Bloco V da UTFPR, Câmpus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Beatriz Terezinha Borsoi (Orientadora), Robison Cris Brito (Convidado) e Rúbia E. O. Schultz Ascari (Convidada), para avaliar o Trabalho de Diplomação da aluna Simone Bampi, matrícula 1147820, sob o título **Sistema Desktop para Realização de Testes de Múltipla Escolha**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, COADS. Após a apresentação a candidata foi entrevistada pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 16:40 hrs foi encerrada a sessão.

Profa. Beatriz Terezinha Borsoi, Dr.
Orientadora

Prof. Robison Cris Brito, M.Sc.
Convidado

Profa. Rúbia E. O. Schultz Ascari, M.Sc.
Convidada

Prof. Omero Francisco Bertol, M.Sc.
Coordenador do Trabalho de Diplomação

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

DEDICATÓRIA

Este trabalho é dedicado a Everaldo meu esposo, que de forma especial e carinhosa me deu força e coragem, me apoiando nos momentos de dificuldades e sendo indispensável nesta jornada. Obrigada por tudo de coração.

AGRADECIMENTOS

Primeiramente agradeço a Deus, por renovar a cada momento a minha força e disposição e pelo discernimento concedido ao longo dessa jornada.

Agradeço também ao meu esposo, que de forma especial e carinhosa me deu força e coragem, me apoiando nos momentos de dificuldades.

Aos meus pais Idalino e Maria de Lourdes que me ensinaram a viver. Pai, contigo aprendi a dedicação ao trabalho ou atividade que me disponho a fazer. Mãe, contigo aprendi a fazer mais de um trabalho ou atividade por vez. Juntos os aprendizados me fazem crescer.

Aos meus colegas de trabalho, que entenderam e me ajudaram na missão de conciliar o trabalho e os estudos.

À minha orientadora, professora Dra. Beatriz T. Borsoi, que ouviu pacientemente as minhas considerações e que sempre me motivou. Quero expressar o meu reconhecimento e admiração pela sua competência profissional e minha gratidão pela sua amizade, por ser uma profissional extremamente qualificada e pela forma humana que conduziu minha orientação.

À professora Ms. Rúbia Eliza de Oliveira Schultz Ascari, pelo conhecimento transmitido e pela disponibilidade em auxiliar nas dificuldades e por meio dela me reporto a todos os professores da Universidade Tecnológica Federal do Paraná (UTFPR).

Aos meus colegas de classe, obrigada pela paciência, pelo sorriso, pelo abraço, pela mão que sempre se estendia quando eu precisava. Esta caminhada não seria a mesma sem vocês.

RESUMO

BAMPI, Simone. **Sistema desktop para realização de testes de múltipla escolha**. 2013. 53 f. Monografia (graduação de Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) - Universidade Tecnológica Federal do Paraná. Pato Branco, 2013.

A realização de testes de múltipla escolha é uma forma de estudar e de verificar a aprendizagem de conteúdos. Simulados de provas como de vestibulares, concursos e teste para obtenção de carteira de motorista são exemplos da aplicação prática desse tipo de teste. Um sistema computacional voltado para esse fim, poderia disponibilizar perguntas para serem respondidas e a correção das mesmas, permitindo às pessoas estudar, por meio da realização de provas simuladas. Verificou-se, assim, a possibilidade de desenvolver um sistema para a realização desse tipo de teste. O sistema foi planejado em dois módulos. Um módulo para cadastro das perguntas que foi implementado como trabalho de estágio. E outro módulo para a realização dos testes, implementado como trabalho de conclusão de curso. Para implementação foi utilizada a linguagem Delphi.

Palavras-chave: Banco de questões. Linguagem Delphi. Sistema Desktop. Testes de múltipla escolha.

ABSTRACT

BAMPI, Simone. **Desktop system to multi-choice tests**. 2013. 48 f. Monografia (graduação de Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) - Universidade Tecnológica Federal do Paraná. Pato Branco, 2013.

Multiple-choice tests area a way to study and verify the learning about contents. Simulated as evidence of vestibular, contests and test to obtain a driver's license are examples of the practical application of this type of test. A system to make available questions to be answered and the correction of the same, would allow people to study, by conducting simulated tests. There is thus the possibility of developing a system to perform this type of test. The system was designed in two modules. One of the modules for the registration of the questions that has been implemented as internship work. And the other module for testing, implemented as completion of course work. For implementation we used the Delphi language.

Palavras-chave: Question Data Base. Delphi language. System for multiple-choice tests.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGO

LISTA DE SIGLAS

CORBA	*Common Object Request Broker Architecture*
COTS	*Commercial Off-The-Shelf*
DBC	Desenvolvimento Baseado em Componentes
DER	Diagrama de Entidades e Relacionamentos
DETRAN	Departamento de Trânsito
ENEM	Exame Nacional do Ensino Médio
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
SQL	*Structured Query Language*
SysML	*Systems Modeling Language*
UML	*Unified Modeling Language*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo são apresentadas as considerações iniciais, com o contexto no qual se insere a proposta deste trabalho, que é um sistema para realização de testes de múltipla escolha. Também são apresentados os objetivos e a justificativa do trabalho. Por fim está a organização do texto por meio da apresentação dos seus capítulos.

1.1 CONSIDERAÇÕES INICIAIS

Testes de múltipla escolha possuem uma aplicabilidade bastante ampla. Eles são de fácil correção por sistemas automatizados. E a resposta às questões é facilitada pela possibilidade de escolha entre alternativas.

Verificou-se, assim, a possibilidade de implementar um aplicativo que disponibilizasse testes de múltipla escolha dos mais diversos assuntos ou categorias. Esse sistema consistiria em cadastro das perguntas, das respectivas alternativas e da alternativa correta. As perguntas estariam organizadas em categorias e agrupadas por níveis de dificuldade. Os testes seriam compostos por escolha aleatória a partir do banco de perguntas.

Os bancos de perguntas cadastrados podem estar relacionados a aplicações ou áreas específicas, como, por exemplo, o Departamento Estadual de Trânsito (DETRAN) ou simulados do Exame Nacional do Ensino Médio (ENEM).

Muitas pessoas podem beneficiar-se de um sistema que prove acesso facilitado ao estudo de determinados conteúdos. Assim, esse trabalho apresenta um software que implementa um banco de perguntas e permite a composição dos testes. O aplicativo, desenvolvido na linguagem Delphi, será dividido em dois módulos: um para cadastro dos testes e outro para realizar os testes. O módulo para cadastro dos testes possibilitará o cadastro das perguntas. O módulo para realizar os testes permitirá ao usuário responder os testes.

1.2 OBJETIVOS

O objetivo geral se refere à finalidade principal de realização deste trabalho. Os objetivos específicos complementam o objetivo geral, no sentido de explicitar os demais resultados obtidos.

1.2.1 Objetivo Geral

Implementar um sistema para realização de testes de múltipla escolha.

1.2.2 Objetivos Específicos

Prover uma maneira facilitada de pessoas estudarem e testarem os seus conhecimentos em determinados conteúdos por meio de testes de múltipla escolha.

Apresentar uma forma de programação *desktop* utilizando formulário padrão e conceitos de orientação a objetos na linguagem Delphi.

1.3 JUSTIFICATIVA

A justificativa do desenvolvimento de um sistema para cadastro de bancos de testes de múltipla escolha e aplicação desses testes deve-se pela ampla aplicabilidade que esse tipo de aplicativo teria para estudo de provas como as aplicadas pelo DETRAN e para estudo para concursos, por exemplo.

O sistema foi implementado de maneira que o usuário possa fazer *download* do instalador que contém um determinado tipo de teste. Para a composição de um teste, a escolha das perguntas será aleatória e realizada pelo próprio aplicativo, seguindo uma lógica estabelecida para que o teste tenha mais perguntas na(s) categoria(s) que houve maior número de erros em teste anterior realizado pelo usuário.

Neste trabalho o ambiente de desenvolvimento e linguagem Delphi na sua versão 7 são utilizados para o desenvolvimento dos dois módulos do sistema. O Delphi 7 fornece

diversas tecnologias integradas visando facilitar o desenvolvimento e aumentar a produtividade do programador.

1.4 ESTRUTURA DO TRABALHO

Este texto está organizado em capítulos, dos quais este é o primeiro, apresentando as considerações iniciais referentes ao trabalho, os objetivos e a justificativa.

O Capítulo 2 contém o referencial teórico do trabalho que está centrado no desenvolvimento de aplicações para *desktop*, programação baseada em eventos e componentes.

No Capítulo 3 estão os materiais e o método utilizado no desenvolvimento deste trabalho.

O Capítulo 4 apresenta a modelagem do sistema e a implementação do módulo *desktop*.

No Capítulo 5 está a conclusão com as considerações finais do trabalho.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta o referencial teórico do trabalho e está centrado no desenvolvimento de aplicações *desktop* e baseadas em eventos e componentes. O sistema desenvolvido é para ambiente *desktop* e implementado utilizando a linguagem Delphi. O desenvolvimento utilizando essa linguagem é realizado a partir de componentes e com a manipulação de eventos desses componentes.

2.1 DESENVOLVIMENTO DE SISTEMAS DESKTOP

Atualmente, o desenvolvimento de aplicativos está bastante direcionado para ambiente *web*. A facilidade e abrangência de acesso da Internet, a simplicidade do cliente (basicamente um navegador *web* instalado) e a possibilidade de uso em redes corporativas (intranets) são alguns dos fatores que colaboram para enfatizar o desenvolvimento de aplicativos para esse tipo de ambiente. Contudo, o processo de escolha de linguagem, tecnologia e mesmo ambiente para desenvolvimento de aplicações para o mercado corporativo pode não ser tão simples e direto.

A grande variedade de linguagens de programação, de gerenciadores de banco de dados e de estilos de interface com o usuário (MACORATTI, 2013) dificulta a escolha de tecnologias, sejam elas para desenvolvimento da interface, da lógica de negócio ou de banco de dados.

Na escolha da tecnologia de desenvolvimento, o tamanho da corporação (empresa de grande ou pequeno porte) para a qual será o aplicativo, a complexidade e a criticidade da aplicação, a infraestrutura existente no cliente (e que ele está disposto a ter), o conhecimento e a experiência da equipe de desenvolvimento são alguns dos fatores que devem ser considerados. Campos (2013) ressalta que a escolha da tecnologia é uma questão de projeto e a avaliação deve ser feita com base nas reais necessidades da empresa e do próprio projeto.

O volume de dados manipulados, a necessidade de integração com outros sistemas e a capacidade de investimento do cliente são aspectos a serem considerados na escolha do banco de dados.

Para a escolha da interface com o usuário, Macoratti (2013) sugere avaliar o tipo de aplicação e indica:

a) Interface *web* – mais adequadas para aplicações voltadas para *e-commerce*, portal, site para uma empresa na Internet, usuários remotos com acesso a Internet e aplicação distribuída entre usuários remotos.

b) Interface *desktop* - para aplicações gráficas como processadores de textos, planilhas eletrônicas, jogos, aplicação cliente/servidor (duas camadas) e aplicações com integração entre vários tipos de hardware (câmeras, scanners, etc.).

A interface *web* apresenta as seguintes vantagens (MACORATTI, 2013): uso de HTML (*HyperText Markup Language*) que é reconhecida por um grande número de usuários já acostumados com o funcionamento dos navegadores; desenvolvimento, manutenção e atualização centralizada da aplicação; a exportação de dados entre usuários remotos usando o protocolo HTTP (*Hypertext Transfer Protocol*) é muito mais fácil do que usar outro protocolo; escalabilidade no processamento, se houver necessidade de aumentar a capacidade de processamento, basta realizar alterações no servidor.

Macoratti (2013) também indica algumas desvantagens das interfaces *web*: a interface HTML pode ser um problema, pois não há uma padronização entre os diversos navegadores e sua aplicação poderia ser exibida de uma maneira diferente dependendo do navegador; a entrada de uma grande massa de dados é prejudicada na interface HTML, pois não existe uma maneira padrão de criar máscaras de entrada de dados; a interface HTML não é rica em controles gráficos e peca no quesito posicionamento; a integração com outros componentes não é tão fácil com HTML; os aplicativos para *web* dependem dos recursos do navegador usado para visualizar a aplicação.

Ressalta-se que as restrições em termos de elementos para compor uma interface *web* utilizando HTML podem ser minimizadas quando ocorre o uso de tecnologias que possibilitem o desenvolvimento de uma aplicação Internet rica, as denominadas *Rich Internet Application*. Tecnologias como *JavaServer Faces*, para citar uma delas. Utilizando bibliotecas de componentes como, por exemplo, *PrimeFaces*, *RichFaces* ou *IceFaces*, permitem o desenvolvimento de aplicativos web com os mesmos padrões de interface que os aplicativos *desktop*.

A interface das aplicações *desktop* apresenta como vantagens (MACORATTI,2013): rica variedade de controles para interface com o usuário; controle sobre o posicionamento dos controles na aplicação; o desempenho para uma interface gráfica é mais rápido em uma aplicação *desktop* que usa o processamento local; criar interface com integração com vários hardware é mais fácil.

Em termos de desvantagens das aplicações *desktop* são citadas (MACORATTI, 2013): uma interface gráfica muito carregada deixa a aplicação mais pesada; a integração com usuários remotos é mais difícil; a distribuição da aplicação é um fator crítico pela diversidade de máquinas dos clientes; a manutenção e atualização da aplicação também se torna mais trabalhosa.

Outro aspecto a ser considerado é se é necessário acesso para o público externo. Uma aplicação híbrida em que um ambiente *desktop* é utilizado para atender a demanda interna e um ambiente *web* é utilizado para publicar os dados (CAMPOS, 2013) pode ser uma solução interessante quando se tem sistemas legados já desenvolvidos para *desktop* ou é necessário proteger dados, por exemplo.

2.2 PROGRAMAÇÃO BASEADA EM COMPONENTES

O Desenvolvimento Baseado em Componentes (DBC) é uma metodologia na qual o desenvolvimento de software ocorre a partir de componentes pré-existentes, ou seja, componentes reutilizáveis.

Marques, Pedroso e Figueira (2013) definem um componente como uma unidade reutilizável de software, tipicamente, com fronteiras bem definidas, sendo encapsulado em um invólucro binário. Esse conceito se refere a componente como o reuso de código e é desta forma que ocorre a atuação das linguagens baseadas no desenvolvimento de componentes

O DBC possibilita a integração planejada de componentes pré-existentes. Essa técnica fornece conjuntos de procedimentos, ferramentas e notações que possibilitam que ao longo do processo de software ocorra tanto a produção de novos componentes quanto a reutilização de componentes existentes (MOURA, CARVALHO, SILVA, 2006).

O recente interesse em DBC está relacionado com a maturidade das tecnologias que permitem a construção de componentes e a combinação destas para o desenvolvimento de aplicações (SPAGNOLI, BECKER, 2003).

É possível agrupar componentes para compor um sistema. Esse agrupamento pode ser realizado com o auxílio de *frameworks* ou com código escrito para essa finalidade. Existem diversas tecnologias (linguagens de programação) que possuem ambientes de desenvolvimento que facilitam a implementação de sistemas com o uso de componentes. Essas ferramentas disponibilizam os componentes por meio de paletas e facilitam a edição das

propriedades dos mesmos e de escrita de código para interagir com os eventos desses componentes.

O processo de desenvolvimento baseado em componentes deve ser caracterizado de modo que qualifique a interface de cada componente, adapte os componentes para remover discordâncias arquiteturais, defina os componentes em um estilo arquitetural selecionado e os atualize, à medida que os requisitos do sistema se modificam (PRESSMAN, 2006).

O DBC considera dois aspectos da arquitetura (D'SOUZA, WILLS, 1998): arquitetura de aplicação e arquitetura técnica. A arquitetura de aplicação preocupa-se com os componentes do domínio, pois ela representa um conjunto de componentes de software, suas relações estruturais (associações e herança entre interfaces e especificações de componentes e relações de composição entre componentes) e dependências de comportamento (relações de dependência entre diferentes componentes, estes e suas interfaces, e entre interfaces). Os componentes dessa arquitetura possuem definição lógica e independente de tecnologia. Já a arquitetura técnica é dependente de tecnologia.

Blois (2006) cita que há muitas referências que justificam que uma arquitetura de componentes deve ser organizada em diferentes camadas. Nessas camadas os componentes possuem diferentes níveis de abstração, e as camadas inferiores prestam serviços para as camadas superiores.

Uma organização das camadas poderia ser: negócio, utilitários e de infraestrutura. Na camada de componentes de negócio são implementados os conceitos ou processos de negócio. Os componentes da camada utilitários prestam serviços genéricos necessários ao desenvolvimento das aplicações. Esses componentes atendem aos interesses dos componentes da camada de negócio e se utilizam de serviços fornecidos pelos componentes da camada de infra-estrutura. E na camada de infraestrutura estão os componentes de infraestrutura, que são responsáveis por estabelecer a comunicação com as plataformas de execução do software. Esses componentes estão mais diretamente vinculados às questões de tecnologia de desenvolvimento.

Os componentes que podem ser adquiridos de fornecedores são os denominados COTS (*Commercial Off-The-Shelf*). Contudo os componentes podem ser disponibilizados juntamente com ambientes de desenvolvimento de linguagens de programação ou ser construídos pela própria empresa ou equipe de projeto.

Associado à utilização de componentes, existem, tipicamente, ambientes visuais que permitem manipulá-los diretamente, quase sem necessidade de codificação. Neste tipo de

programação, o código fonte escrito é normalmente uma cola entre os componentes, implementando certa "lógica de negócio" que orquestra as relações e a utilização dos componentes (MARQUES, PEDROSO, FIGUEIRA, 2013).

Do ponto de vista de programação, um componente corresponde a uma classe. No entanto, existem três elementos básicos, muito importantes, que suportam a sua utilização e a interligação a outros componentes (MARQUES, PEDROSO, FIGUEIRA, 2013):

a) Propriedades - uma propriedade representa um aspecto do estado de um componente. Por exemplo, se há um componente que represente um botão na tela, uma propriedade poderia ser o tamanho do botão e outra o seu título. Em termos de programação, uma propriedade é como uma variável pública de um objeto. A diferença é de que existe um método que é chamado quando o seu valor é alterado e outro método que é chamado quando o seu valor é lido. Em termos gerais, os estados de um componente devem ser definidos pelo valor das suas propriedades, como ocorre com objetos.

b) Métodos - os métodos podem ser associados com os métodos de uma classe em orientação a objetos. Quando um método de um componente é acessado (chamado) existe uma ação que é realizada nesse método. Os métodos representam ações que não podem ser manipuladas ou realizadas visualmente.

c) Eventos - um evento representa um acontecimento. É uma notificação. Quando um componente envia um evento, os receptores desse evento são notificados. Um componente pode registrar-se com outros componentes para receber eventos e, por sua vez, pode lançar eventos.

2.3 PROGRAMAÇÃO ORIENTADA A EVENTOS

Programas orientados a evento geralmente consistem em tratadores, que são programas que processam os eventos para produzir respostas, e um emissor, que invoca os tratadores. Outra alternativa consiste em disparar os tratadores por eles próprios, criando um efeito de evento em cascata (SILVEIRA, VARGAS, 2007).

Os eventos são funções executadas em um determinado momento, dependendo de seu tipo. Por exemplo, o evento *FormShow* de um formulário é acionado quando o formulário é executado/apresentado; o evento *MouseDown* é executado quanto o botão é pressionado e *MouseUp* quando o botão do mouse é liberado. Cada componente tem seus eventos, embora

haja eventos que possuem nome e mesmo funcionalidades semelhantes em componentes distintos. Em uma linguagem de programação orientada a eventos, todas as ações que ocorrem durante a execução do programa são estruturadas nos eventos dos objetos. Por exemplo: se há um Botão denominado "Button1" e o usuário clicar sobre o mesmo será acionado o evento Button1Click do Button1. E uma ação (método) poderá estar associada a esse evento, no sentido de que seja realizada alguma ação quando esse evento ocorre.

Delphi é uma linguagem de programação orientada a eventos, ou seja, quando é executada uma ação o programa executa outra ação pré-determinada. Essa linguagem de programação também utiliza o paradigma de orientação a objetos que fornece suporte a abstração de dados.

3 MATERIAIS E MÉTODO

Este capítulo apresenta os materiais e o método utilizados. Os materiais se referem às tecnologias e ferramentas utilizadas para modelar e implementar o sistema. O método reporta a sequência das principais atividades realizadas para desenvolver este trabalho.

3.1 MATERIAIS

Foram utilizadas as seguintes tecnologias e ferramentas para a modelagem e a implementação do sistema:

a) Visual Paradigm for UML – para a modelagem do sistema;

b) Delphi 7 – como IDE (*Integrated Development Environment*) de desenvolvimento;

c) Firebird versão 2.5 – para o banco de dados;

d) IBExpert – para interação com o banco de dados;

e) FastReport – para a geração dos relatórios.

3.1.1 Visual Paradigm

Visual Paradigm for UML (VP-UML) é uma ferramenta de modelagem que permite definir todos os tipos de diagramas UML (*Unified Modeling Language*) (VISUAL PARADIGM, 2012), além de diagramas de entidades e relacionamentos e mapeamento objeto relacional. Visual Paradigm for UML suporta três fases de modelagem de dados com Diagrama de Entidades e Relacionamentos (DER) conceitual, lógico e físico.

Essa ferramenta fornece suporte para gerenciamento de casos de uso, diagrama de requisitos SysML (*Systems Modeling Language*) e projeto de banco de dados com diagrama de entidades e relacionamentos. A Figura 4 apresenta a interface principal dessa ferramenta.

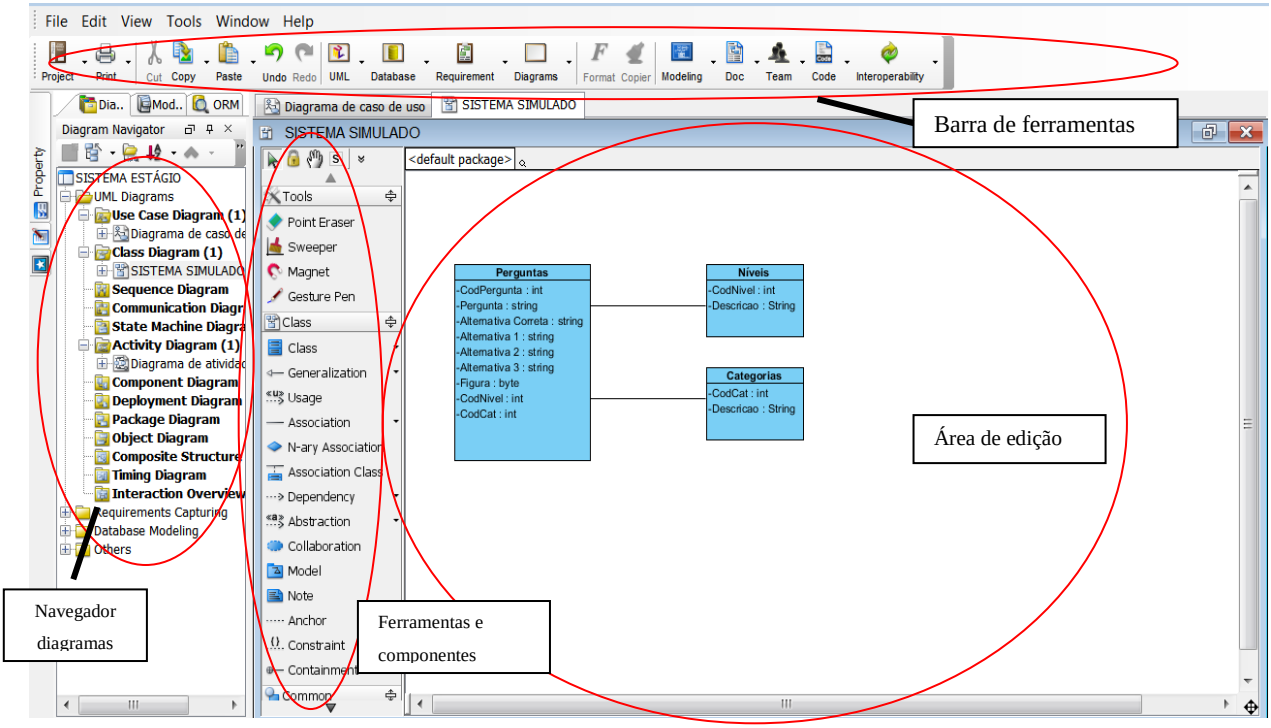


Figura 1 – Tela inicial da ferramenta Visual Paradigm

As partes destacadas na Figura 1 apresentam as principais funcionalidades da ferramenta Visual Paradigm e são:

a) Navegador de diagramas – nesta área da interface são apresentados os diagramas armazenados. Os arquivos são apresentados por pastas e por tipo de diagrama, facilitando a localização dos mesmos e organização do projeto.

b) Propriedades – apresenta as propriedades do elemento selecionado do diagrama que está em edição.

c) Área de edição – nesta área são criados graficamente os diagramas por meio dos elementos dispostos na barra que fica na lateral esquerda dessa área.

d) Ferramentas e componentes – área que contém os componentes utilizados para compor o diagrama e as ferramentas relacionadas ao diagrama em edição.

e) Barra de ferramentas – com os atalhos para as principais funcionalidades do Visual Paradigm.

A ferramenta Visual Paradigm produz diagramas compatíveis com a UML 2.1 e possibilita a geração de código na linguagem Java a partir de diagramas. Além de engenharia reversa em Java, C++, XML (*Extensible Markup Language*) Schema, .Net e CORBA IDL (*Common Object Request Broker Architecture Interactive Data Language*). Além de permitir a geração de códigos compatíveis com XML.

3.1.2 Delphi

Delphi é uma linguagem de programação de alto nível e é baseado na primeira linguagem Object Pascal. Vantagens da IDE Delphi (CANTU, 2003):

- Muito utilizado para desenvolvimento de aplicações multicamadas e cliente/servidor.
- Permite acesso aos bancos de dados mais conhecidos do mercado.
- Construtor visual de interface com o usuário baseado em formulários e componentes.
- Arquitetura baseada em componentes (reutilização e manutenção).
- Compilador de código nativo.
- *Drag-and-Drop Design* permitindo que o código seja gerado automaticamente durante a montagem do formulário.
- *Tow-Way Tools* que permite alternar entre um formulário e o seu código (*unit*).
- Biblioteca de componentes visuais.
- *Visual Component Library* que consiste de objetos reutilizáveis incluindo objetos padrão de interface com o usuário, gerenciamento de dados, gráficos e multimídia, gerenciamento de arquivos.

Delphi possui uma arquitetura aberta. Possibilita adicionar componentes e ferramentas personalizadas de terceiros.

A Figura 2 apresenta a tela inicial da IDE de desenvolvimento da linguagem Delphi 7. No detalhe desta Figura está a parte superior da tela, destacando-se as barras de títulos, menus e ferramentas, os botões da janela e a paleta de componentes.

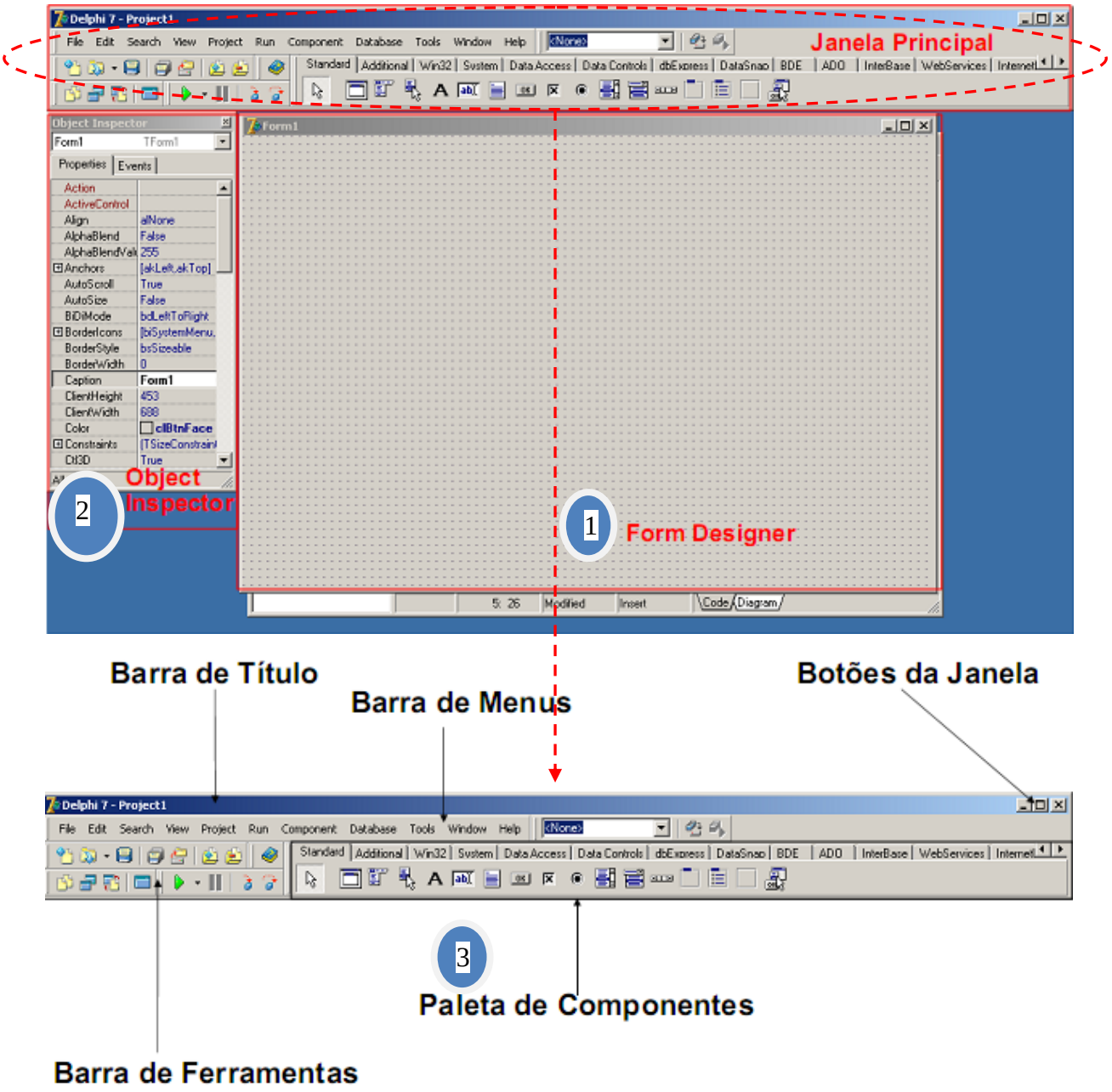


Figura 2 – Tela inicial da IDE Delphi 7

As partes marcadas da Figura 2:

1. *Form Design* - é a janela que contém a interface gráfica da aplicação em desenvolvimento. No *form* são inseridos os componentes que farão parte da interface com o usuário.
2. *Object Inspector* - define as propriedades e os eventos para os componentes. Propriedades são características do componente selecionado e eventos são os eventos suportados pelo componente selecionado.

3. Paleta de Componentes - é a barra de ferramentas com os componentes para serem utilizados no desenvolvimento de aplicações. A paleta é constituída de várias guias. Cada guia possui componentes agrupados por finalidade.

3.1.3 Firebird

Firebird é um banco de dados relacional que oferece muitas características padrão Ansi SQL (*Structured Query Language*) que funciona em Linux, Windows, Mac OS e uma variedade de plataformas Unix (FIREBIRD, 2012).

A Fundação FirebirdSQL coordena a manutenção e desenvolvimento do Firebird, sendo que os códigos fonte são disponibilizados sob o CVS da SourceForge.

Firebird oferece concorrência, de alto desempenho e apoio de uma linguagem para procedimentos armazenados e gatilhos.

O Firebird é gratuito em todos os sentidos: não há limitações de uso, e seu suporte amplamente discutido em listas na Internet, o que facilita enormemente a obtenção de ajuda técnica.

O produto é bastante seguro e confiável, suportando sistemas com centenas de usuários simultâneos e bases de dados com dezenas/centenas de gigabytes. Há suporte gratuito na Internet através de vários sítios.

O Firebird é amplamente utilizado em todo o mundo, com a maior base de usuários no Brasil, Rússia e Europa.

3.1.4 IBExpert

O IBExpert (IBEXPERT, 2012) é um ferramenta que possibilita o gerenciamento de bases de dados de Interbase e Firebird em diversas versões. O IBExpert inclui editores visuais para todos os objetos de banco de dados, um SQL Editor e gerador de *script*, um depurador para procedimentos armazenados e gatilhos, um Query Builder, um Designer de banco de dados e a sua própria linguagem, o IBEBlock.

Seus bancos de dados e seus objetos são apresentados em uma estrutura de árvore.

Possui opções como arquivos de log, backup e restore de arquivos, caminhos padrão, filtros, scripts, operações.

Com as Strodes Procedures e as Triggers é possivel:

- personalizar modelos.
- criar um procedimento a partir de uma consulta ou diretamente do Tabela e Field Editores.
- Executar debug.
- Analisar o desempenho.

O IBExpert tem ainda funções como:

- seletividade Recompute de todos os índices.
- Recompilar todas as triggers.
- Monitoramento de banco de dados.
- Propriedades do Banco de Dados.
- SQL monitor.
- Log Manager.
- Localize IBExpert.
- Recompilar todos os procedimentos armazenados (stored procedures).
- Validação de Banco de Dados.
- Banco de Dados Shutdown (encerramento).
- Banco de Dados On-line.
- Propriedades do servidor / log.
- Localizar IB Mensagens.

3.1.5 FastReport

FastReport é um gerador de relatórios para todas as versões de Delphi, incluídas as versões 4 e 7. FastReport é um módulo adicional que permite sua aplicação gerar relatórios de forma rápida e eficiente (FAST REPORT, 2013)..

O FastReport 4, tem cinco edições com diferentes características:

- Embarcadero RAD Edition: Esta edição é distribuída com o Embarcadero RAD Studio XE 2.
- Basic Edition: Atende às necessidades de desenvolvedores que estão criando aplicativos com relatórios diretos de lógica.

- Standard Edition: O Standard Edition inclui: Um designer de diálogo. Um mecanismo de *script* interno, que permite que você manipule a lógica dos relatórios, que suporta quatro idiomas (PascalScript, C + + Script, JScript, e BasicScript).
- Professional Edition: Além de todas as características do FastReport 4 Standard Edition, o Professional Edition inclui também a *built-in SQL Query Builder*, que permite a criação de consultas complexas sem ter um bom domínio de SQL, códigos fonte completo do gerador de relatórios, que permite compreender completamente a lógica de trabalho e fazer as alterações necessárias e ajustes menores.
- Enterprise Edition:Além de todas as vantagens de FastReport 4 Professional Edition, esta edição inclui componentes web-relatórios.

3.2 MÉTODO

Para o desenvolvimento deste trabalho algumas etapas foram realizadas. A seguir essas etapas estão descritas juntamente como as suas principais atividades.

a) Definição inicial do escopo do sistema a ser desenvolvido

Definiu-se que o sistema estaria dividido em dois módulos. Um deles com acesso pelo usuário administrador e que permitiria os cadastros relacionados à composição do teste. O outro com acesso aos usuários que realizariam os testes.

b) Definição das tecnologias a serem utilizadas

Com a definição do sistema a ser desenvolvido, foi necessário escolher as ferramentas que atendessem os requisitos para o desenvolvimento da aplicação. Escolheu-se a linguagem Delphi 7 pelos recursos que a mesma oferece. Após definidas as tecnologias e as ferramentas, foi necessário instalá-las e configurá-las para se iniciar o desenvolvimento.

c) Estudo das tecnologias

O conhecimento das tecnologias foi adquirido com orientações de professores e de forma autodidata por meio de pesquisas, exemplos, tutoriais, aplicações modelo.

d) Requisitos

Os requisitos foram definidos com uma listagem de requisitos funcionais e não funcionais. Em seguida eles foram modelados sob a forma de um diagrama de casos de uso e descrições suplementares. A definição dos requisitos teve como base as necessidades de uma pessoa que estuda para realizar um teste do DETRAN. Percebeu-se que essas necessidades poderiam ser extrapoladas para a realização de qualquer tipo de teste que envolvesse questões de múltipla escolha.

e) Análise e Projeto

Na análise e projeto foram definidos os casos de uso, a modelagem das classes e das tabelas do banco de dados

f) Codificação

A implementação foi realizada utilizando a linguagem Delphi e por meio do ambiente de desenvolvimento vinculado à própria linguagem.

g) Testes

Os testes realizados tiveram o único objetivo de identificar erros de código e foram realizados pela autora deste trabalho.

4 SISTEMA DESENVOLVIDO

Este capítulo apresenta um sistema desenvolvido como resultado deste trabalho. Da modelagem estão os casos de uso, diagramas de classes e de entidades e relacionamentos do banco de dados. Da implementação estão exemplos da codificação realizada. O sistema é apresentado pela explicação da interação com o mesmo por meio da sua interface.

4.1 APRESENTAÇÃO DO SISTEMA

O aplicativo desenvolvido é um sistema computacional para armazenamento de questões de múltipla escolha e suas respostas. As questões armazenadas possuem categoria e nível de dificuldade associado. Os testes, como uma composição de questões, ficam armazenados no sistema.

O sistema pode ser usado para treinamento dos testes do DETRAN, do ENEM ou outros definidos pelos usuários. As perguntas, bem como suas alternativas, incluindo a indicação da resposta correta serão cadastradas em um banco de dados.

A definição das perguntas que irão compor os testes será realizada no momento que o usuário iniciar o teste. A escolha das perguntas que comporão um teste será realizada de maneira aleatória a partir das perguntas armazenadas do banco de dados.

Para a composição do teste foram estabelecidos os seguintes critérios:

a) Se é o primeiro teste realizado pelo usuário

O teste é composto por quantidade igual de perguntas para cada uma das categorias cadastradas.

Para determinar a quantidade de perguntas de cada categoria que compõe o teste:

Quantidade de perguntas por categoria = (Total de perguntas do teste / quantidade de categorias)

Para compor o teste:

Escolher aleatoriamente do banco de dados a quantidade de perguntas para cada categoria.

b) Se não é o primeiro teste realizado pelo usuário

Ordenar a quantidade de erros por categorias de perguntas do teste anterior. A(s) categoria(s) que tiver(em) maior quantidade de erros recebe(m) três perguntas. A quantidade

restante de perguntas para compor o teste é dividida entre a quantidade de categorias existentes.

4.2 MODELAGEM DO SISTEMA

O Quadro 1 apresenta os requisitos funcionais identificados para o sistema.

Identificação	Nome	Descrição
RF001	Criar novo banco de dados	Criar novo banco de dados composto por perguntas relacionadas a um conteúdo de interesse.
RF002	Excluir perguntas do banco de dados.	Excluir perguntas cadastradas, realizado pelo módulo cadastro do sistema.
RF003	Cadastrar perguntas no banco de dados	Incluir perguntas no banco de dados.
RF004	Alterar pergunta no banco de dados	Realizar alterações em perguntas cadastradas, realizado pelo módulo cadastro do sistema.
RF005	Realizar teste.	O usuário terá perguntas aleatórias para responder e ao finalizar o teste receberá o resultado do mesmo.
RF006	Registrar total de acertos e erros de cada teste.	O sistema gerará um relatório ao final do teste, especificando a quantidade de acertos e erros.

Quadro 1 – Requisitos funcionais

No Quadro 2 estão os requisitos não-funcionais identificados para o sistema. Os requisitos não funcionais explicitam regras de negócio, restrições de acesso ao sistema, por exemplo, requisitos de qualidade, desempenho, segurança e outros.

Identificação	Nome	Descrição
RNF01	Diversificar os bancos de dados	O sistema poderá ter perguntas de diversas áreas. Isso será realizado por meio do cadastramento de bancos de dados distintos.
RNF02	Colorir respostas	Colorir a resposta de cada questão (verde) que for a correta sendo que se ela não foi a resposta do usuário sua resposta aparecerá em outra cor (vermelho).

Quadro 2 – Requisitos não funcionais

Os casos de uso representam agrupamentos de requisitos funcionais. A Figura 3 apresenta o diagrama de caso de uso do sistema.

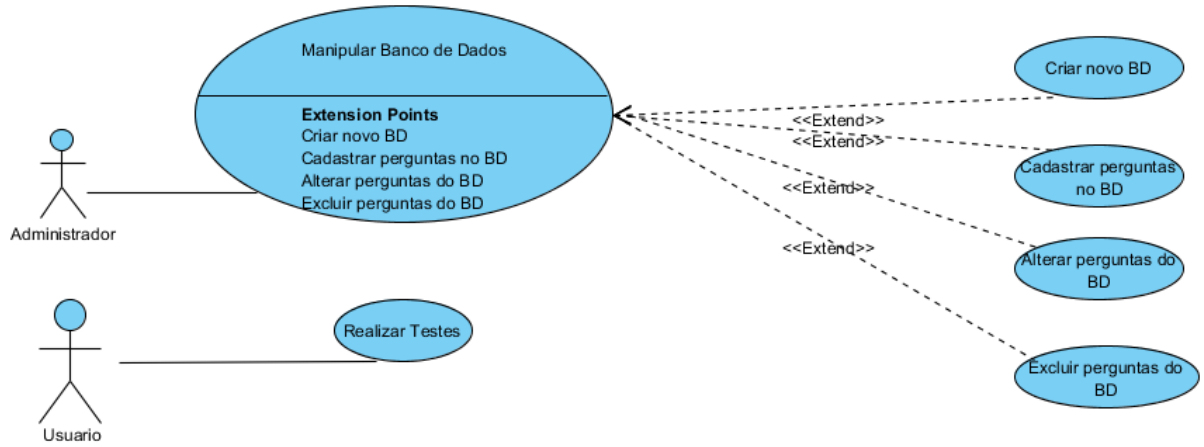


Figura 3 – Diagrama de casos de uso

Como apresentado na Figura 3, foram definidos dois atores para o sistema. O ator Usuário iniciará o teste e responderá as perguntas. O ator Administrador será encarregado de manipular os bancos de dados disponíveis em um dado local da rede, gerenciar (inclusão, exclusão e alteração) os dados do banco, bem como cadastrar, alterar e excluir perguntas e respostas.

O Quadro 3 resume os requisitos definidos para o sistema. Esses requisitos estão sob a forma de descrição dos casos de uso apresentados na Figura 3.

Identificação	Objetivo	Requisitos que o compõem
Manipular banco de dados	O administrador fará cadastro, alteração e exclusão de perguntas em um banco ou poderá criar um novo banco de dados.	Criar novo banco de dados. Cadastrar perguntas no banco de dados. Alterar pergunta no banco de dados. Excluir pergunta no banco de dados.
Realizar Testes	Permitir que o ator responda perguntas, apresentando ao final o resultado, total de acertos e erros.	Realizar teste com número fixo de perguntas. Registrar total de acertos e erros.

Quadro 3 – Casos de uso

No Quadro 4 está a descrição do caso de uso manipular banco de dados. Nesse caso o administrador pode criar um banco de dados com questões relativas a um determinado assunto, bem como alterar e excluir questões.

Identificador do requisito: Manipular banco de dados.	
Descrição: Este caso de uso permite a manipulação de um banco criado ou mesmo criar um novo banco de questões.	
Evento Iniciador: Tela inicial do servidor.	
Atores: Administrador.	
Pré-condição: Abrir o sistema, ter acesso ao servidor.	
Sequência de Eventos: 1 – Escolher a opção alterar banco ou criar novo. 2 – Fazer as alterações necessárias.	
Pós-Condição: Um novo banco ou um banco com atualizações estará disponível no servidor.	
Extensões:	
Nome do fluxo alternativo (extensão)	Descrição
Alteração interrompida	Se uma alteração for interrompida, ela não será incluída, como por exemplo, uma pergunta sem as respostas necessárias.

Quadro 4 – Caso de uso manipular banco de dados

O Quadro 5 apresenta o caso de uso para realizar os testes.

Identificador do requisito: Realizar testes
Descrição: Este caso de uso permite realizar um teste (responder questões).
Evento Iniciador: Tela de início para realização de um teste.
Atores: Usuário.
Pré-condição: O usuário deverá ter instalado o teste.
Sequência de Eventos: 1 – Iniciar teste. 2 – Responder as questões. 3 – Visualizar o resultado do teste realizado.
Pós-Condição: Apresentar a quantidade de erros e acertos.

Quadro 5 – Casos de uso realizar testes

A Figura 4 apresenta as classes definidas para o sistema. Nos Quadros 6 a 9 está a descrição dessas classes.

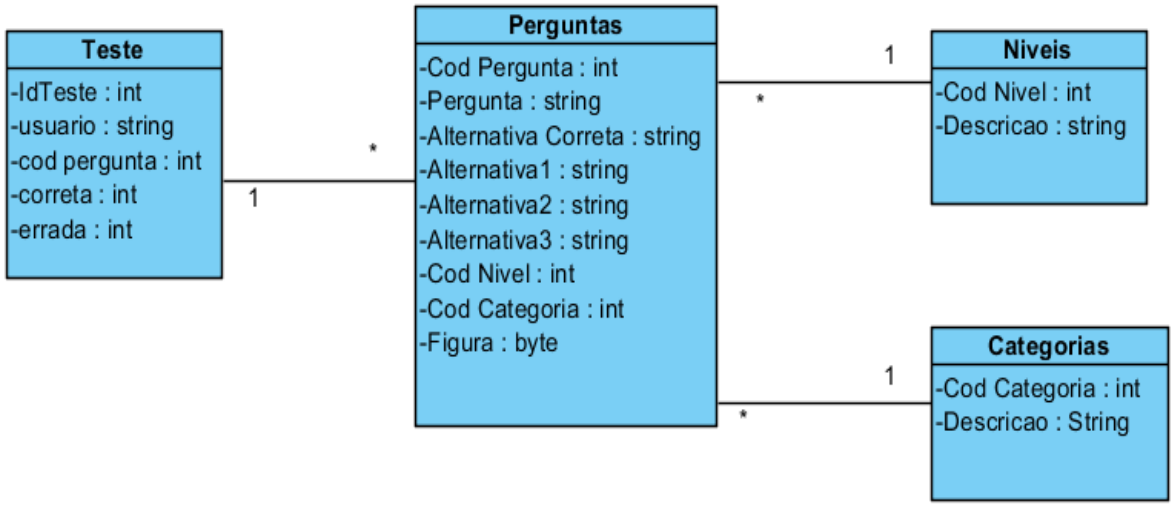


Figura 4 – Diagrama de classes

No Quadro 6 está a descrição da classe Perguntas.

Identificação:	**Perguntas**
Descrição:	Uma pergunta se refere a um assunto especifico e a uma categoria. Cada pergunta possui uma alternativa como a resposta correta e mais três alternativas incorretas, além de uma figura que é de cadastro opcional.
Atributos:	Pergunta (String): Pergunta Alternativa Correta (String): resposta certa da pergunta. Alternativa 1 (String): resposta errada da pergunta Alternativa 2 (String): resposta errada da pergunta Alternativa 3 (String): resposta errada da pergunta Figura (Picture): se houver figura, será correspondente a pergunta.
Métodos:	Cadastrar, Alterar, Consultar e Excluir

Quadro 6 – Classe Perguntas

O Quadro 7 apresenta a descrição da classe Níveis.

Identificação:	**Níveis**
Descrição:	Especifica o nível de dificuldade da pergunta.
Atributos:	Descrição (String): Descrição
Métodos:	Cadastrar, Alterar, Consultar e Excluir.

Quadro 7 – Classe Níveis

A descrição da classe Categorias é apresentada no Quadro 8.

Identificação:	**Categorias**
Descrição:	Especifica a categoria da pergunta.
Atributos:	Descrição (String): Descrição
Métodos:	Cadastrar, Alterar, Consultar e Excluir.

Quadro 8 – Classe Categorias

No Quadro 9 está a descrição da classe Teste.

Identificação:	**Teste**
Descrição:	Um teste possui o registro do usuário que esta realizando o teste bem como os acertos e erros obtidos em cada pergunta.
Atributos:	IdTeste (int): código do teste usuario (String): Nome do Usuário Cod. Pergunta (int): Código da Pergunta Correta (Int): contagem das corretas. Errada (int): contagem das erradas.
Métodos:	Consultar.

Quadro 9 – Classe Teste

O diagrama de entidades e relacionamentos do banco de dados está apresentado na Figura 5.

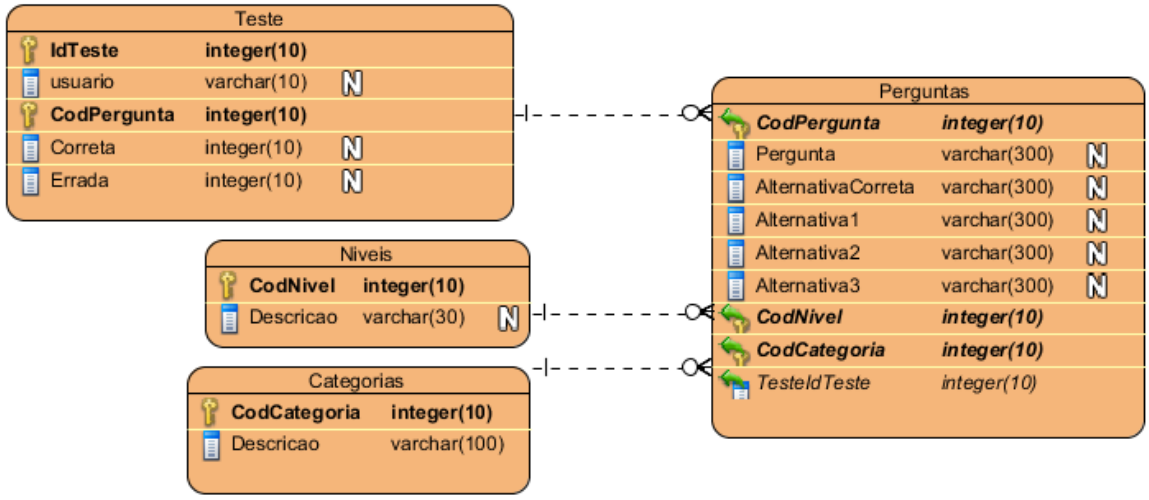


Figura 5 – Diagrama de entidades e relacionamento

Como pode ser visualizado na Figura 5, a tabela Perguntas possui relacionamentos com as tabelas Categorias, Níveis e Teste. Uma pergunta pertence a uma determinada categoria e nível. Muitas Perguntas podem estar em um Teste.

4.3 DESCRIÇÃO DO SISTEMA

A descrição do sistema está dividida em dois módulos conforme seu desenvolvimento.

4.3.1 Descrição do módulo Cadastro

A tela principal do sistema apresenta o menu de cadastros e de relatório. Por meio desse menu o usuário poderá cadastrar categorias, níveis e perguntas, assim como visualizar um relatório de perguntas cadastradas. Nesta opção ainda há o item sair que ao ser escolhido permite sair do sistema. Os itens do menu cadastros estão apresentados na Figura 6.

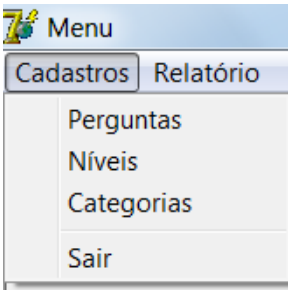


Figura 6 – Opções de cadastros

Na Figura 7 pode ser observada a tela na qual são realizados os cadastros de categorias. Nessa tela há atributos que deverão ser preenchidos pelo usuário como o código da categoria e a descrição da mesma. Ambos os campos devem obrigatoriamente ser preenchidos. No *grid* na parte inferior da tela são apresentados os registros já armazenados no respectivo cadastro.

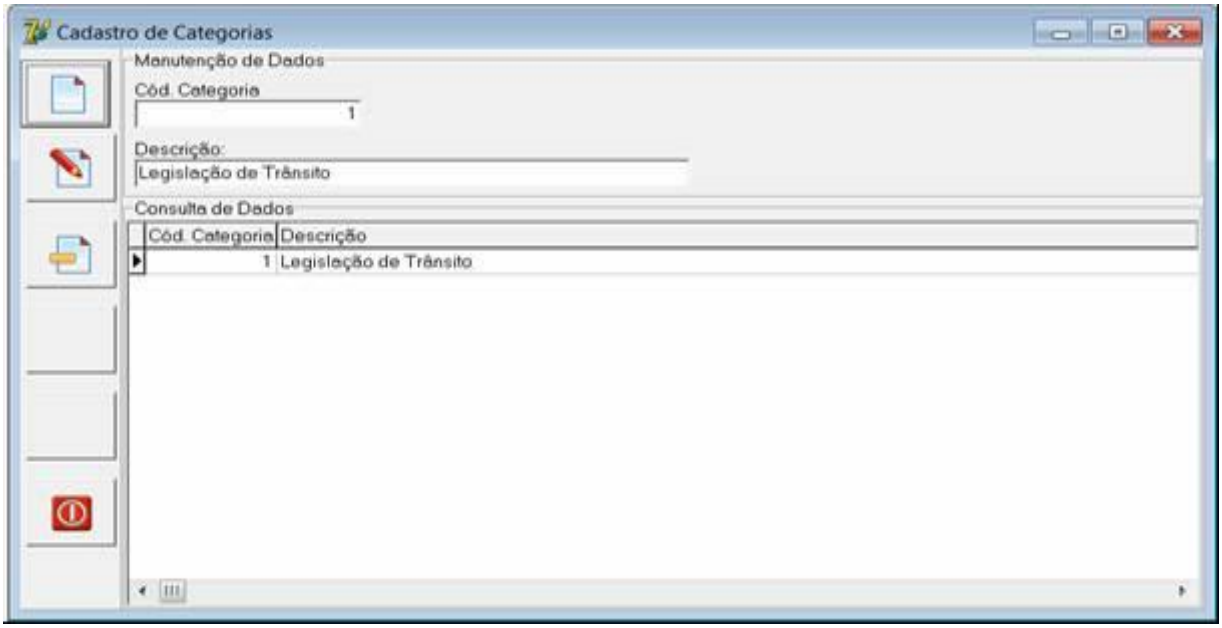


Figura 7 – Tela cadastro de categorias

Na Figura 8 pode ser observada a tela na qual são realizados os cadastros de níveis. Na tela há atributos que deverão ser preenchidos pelo usuário como o código do nível e a descrição do mesmo. Ambos os campos devem obrigatoriamente ser preenchidos. No *grid* na parte inferior da tela pode ser visualizado o que foi cadastrado.

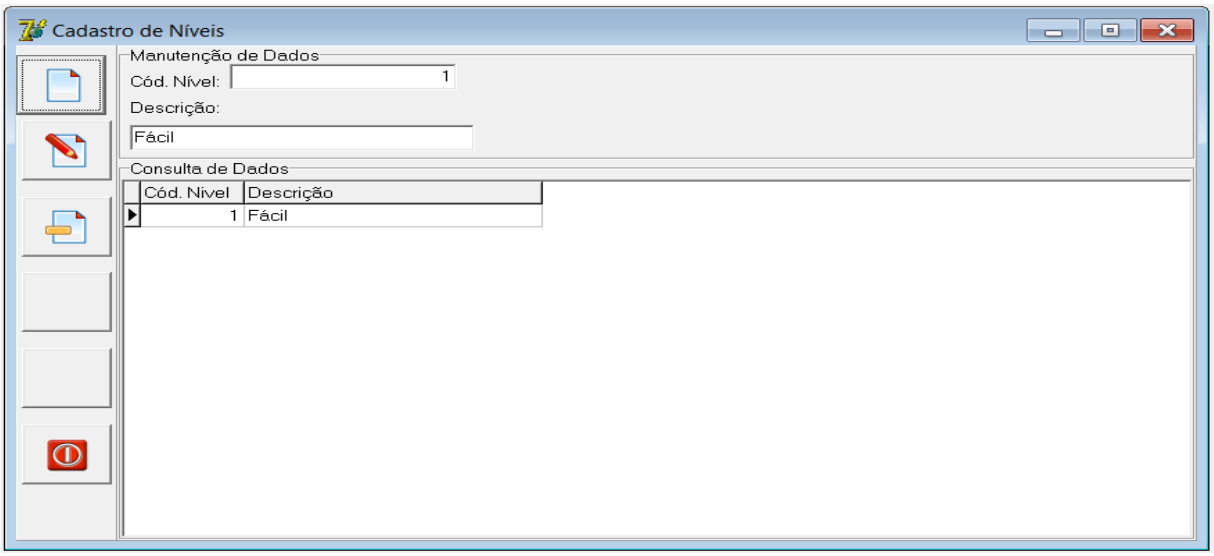


Figura 8 – Tela cadastro de níveis

Na Figura 9 pode ser observada a tela na qual são realizados os cadastros de perguntas. Nessa tela há atributos que deverão ser preenchidos obrigatoriamente pelo usuário como o nível da pergunta, a categoria da pergunta, o seu enunciado, a alternativa correta e outras duas alternativas. A alternativa três não é de preenchimento obrigatório e também é opcional a inserção de imagem.

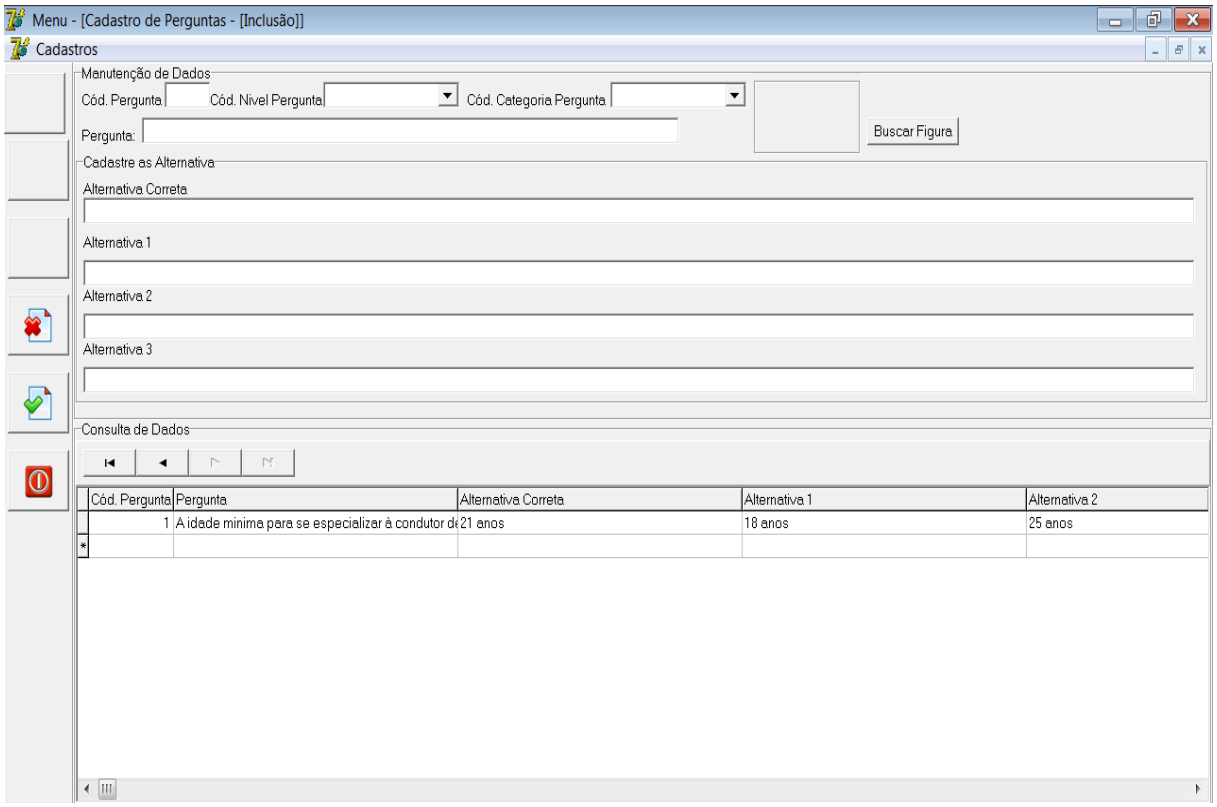


Figura 9 – Tela cadastro Perguntas

No menu de relatórios é possível visualizar as perguntas cadastradas separadas por categoria, conforme Figura 10.

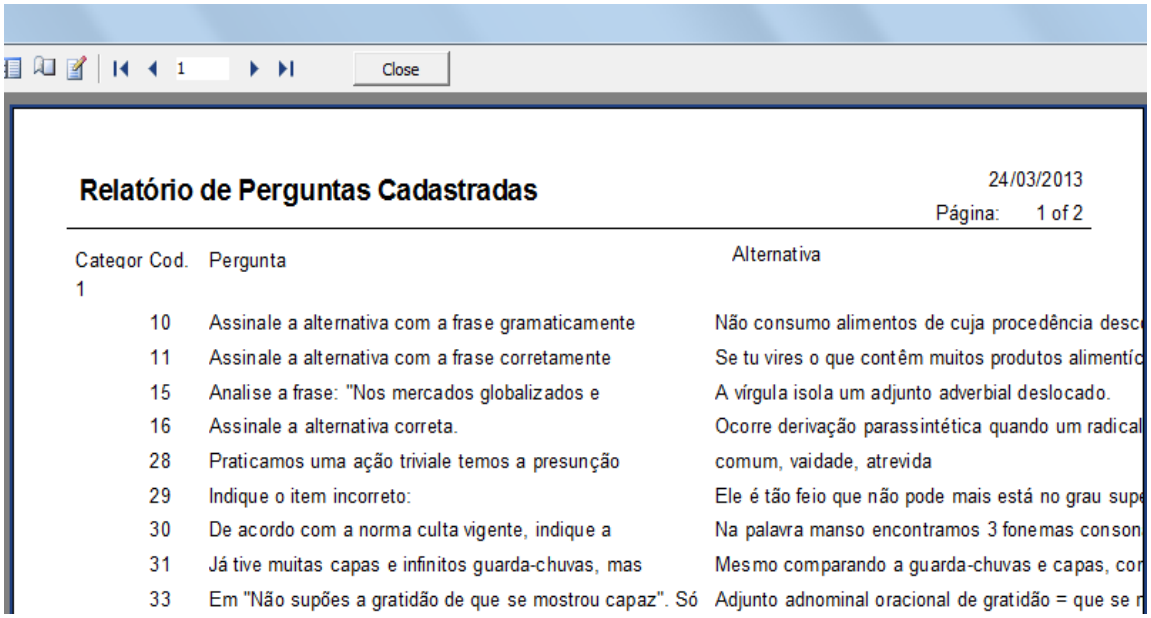


Figura 10 – Relatório de perguntas cadastradas

4.3.2 Descrição do módulo Teste

Para responder ao teste o usuário fará conexão inserindo seu nome. Por meio desse formulário o usuário poderá ter acesso ao menu para responder as perguntas, é neste momento que o usuário é identificado. Nesta opção ainda há o item Fechar que ao ser escolhido permite sair do sistema. Os itens do acesso estão apresentados na Figura 11.

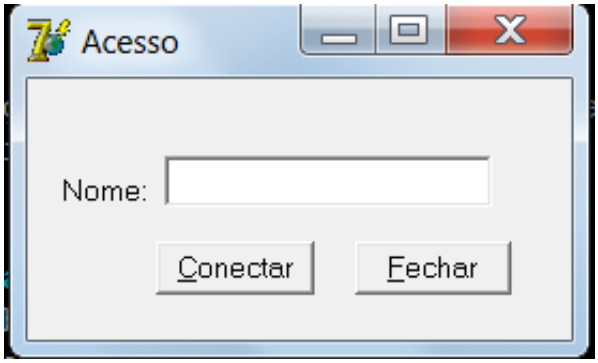


Figura 11 – Formulário de Acesso ao sistema

A tela de Menu do sistema apresenta os ícones “Acessar” e “Relatórios”. Por meio desse menu o usuário poderá responder um novo teste, ou consultar os relatórios de seus testes já efetuados. Nesta opção ainda há o item sair que ao ser escolhido permite sair do sistema. Os itens do menu estão apresentados na Figura 12. É possível verificar também a identificação do usuário na *statusBar* (barra de status, apresentada na parte inferior da tela).

Figura 12 – Menu do sistema

No formulário de Teste, visualizado na Figura 13, é apresentada a pergunta e suas respectivas opções de resposta. Ao responder pode ser verificado o resultado da pergunta. O botão conferir tem a função de verificar se a resposta marcada é a certa, sendo que enquanto o botão conferir não é acionado o botão próximo fica desabilitado. Assim que a pergunta é conferida o botão próximo é habilitado para que possa ser acessada uma nova pergunta.

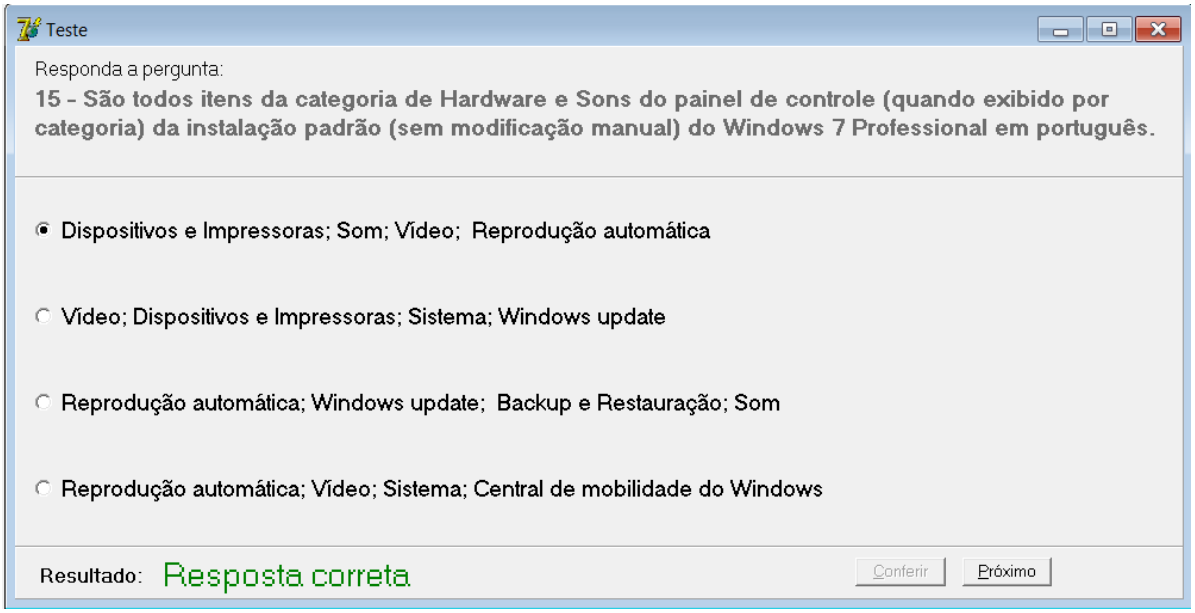
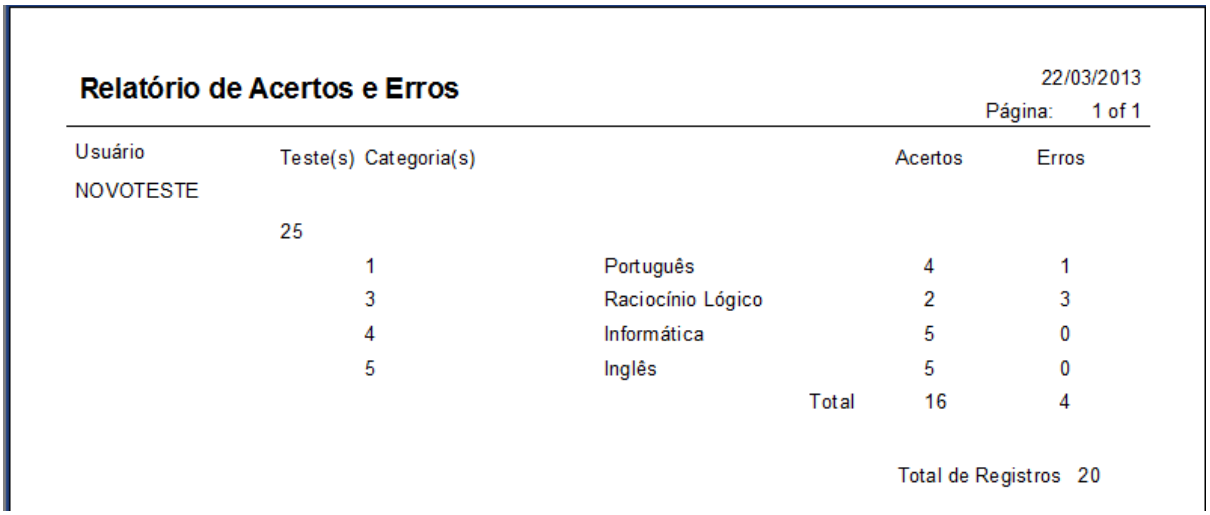


Figura 13 – Formulário Teste

No menu de relatórios é possível visualizar os acertos e erros de cada usuário separados por categoria e por teste, visualizado na Figura 14.



Relatório de Acertos e Erros

22/03/2013
Página: 1 of 1

Usuário	Teste(s)	Categoria(s)		Acertos	Erros
NOVOTESTE					
	25				
	1	Português		4	1
	3	Raciocínio Lógico		2	3
	4	Informática		5	0
	5	Inglês		5	0
			Total	16	4

Total de Registros 20

Figura 14 – Relatório de acertos e erros

4.4 IMPLEMENTAÇÃO DO SISTEMA

Os códigos do sistema são apresentados a seguir organizados nos dois módulos que foram desenvolvidos.

4.4.1 Módulo Cadastro das Perguntas

No módulo de cadastro foram usados os códigos implementados no formulário padrão que permitem inclusão, alteração, exclusão de perguntas como pode ser observados na listagem 1.

```
procedure TfrmPadraoDB.btnNovoClick(Sender: TObject);
begin
  EstadoBotoes(tebIncluir);
  SelectFirst;
  cds.Append;
end;

procedure TfrmPadraoDB.btnAlterarClick(Sender: TObject);
begin
     EstadoBotoes(tebAlterar);
     cds.Edit;
end;

procedure TfrmPadraoDB.btnExcluirClick(Sender: TObject);
begin
  EstadoBotoes(tebExcluir);
  if   (not   cds.IsEmpty)   and   (msgSN('Deseja   excluir   o   registro
selecionado?')= idYes)then
    begin
    EstadoBotoes(tebCancelar);
    cds.Delete;
    if cds.ApplyUpdates(0) <> 0 then
      msgOK('Erro ao excluir o registro!');
    end
    else
      EstadoBotoes(tebCancelar);
end;

procedure TfrmPadraoDB.btnCancelarClick(Sender: TObject);
begin
  EstadoBotoes(tebCancelar);
  cds.Cancel;
end;

procedure TfrmPadraoDB.btnSalvarClick(Sender: TObject);
begin
  if DadosValidos then
    begin
     cds.Post;
     if cds.ApplyUpdates(0) = 0 then
```

```
      EstadoBotoes(tebCancelar)
     else
      msgOK('Erro ao gravar o registro!');
     end;
  end;
```

Listagem 1 – *Procedures* para inclusão, alteração e exclusão

A *procedure* "EstadoBotoes" é responsável por habilitar ou desabilitar os botões da tela, de acordo com a operação a ser realizada, conforme apresentado na Listagem 1.

```
procedure TfrmPadraoDB.EstadoBotoes (Oper: TEstadoBotoes);
begin
 btnNovo.Enabled := (Oper = tebCancelar) or (Oper = tebExcluir);
 btnAlterar.Enabled := btnNovo.Enabled;
 btnExcluir.Enabled := btnNovo.Enabled;
 grbConsulta.Enabled:= btnNovo.Enabled;
 btnSalvar.Enabled := (Oper = tebIncluir) or (Oper = tebAlterar);
 btnCancelar.Enabled := btnSalvar.Enabled;
 TipoManutencao(Oper);
end;
```

Listagem 2 – Procedure TfrmPadraoDB.EstadosBotoes

A *procedure* TfrmPadraoDB.TipoManutencao da Listagem 2 chama a *procedure* "TipoManutencao" apresentada na Listagem 3, de acordo com o botão pressionado. Essa procedure é responsável por ajustar o título do formulário, apresentando ao usuário a operação que está sendo realizada.

```
procedure TfrmPadraoDB.TipoManutencao(Oper: TEstadoBotoes);
begin
 if Pos(' - [', Caption) > 0 then
 Caption := Copy (Caption, 1, Pos(' - [', Caption)-1);

 case Oper of
 tebIncluir: Caption := Caption + ' - [Inclusão]';
 tebAlterar: Caption := Caption + ' - [Alteração]';
 tebExcluir: Caption := Caption + ' - [Exclusão]';
 end;
end;
```

Listagem 3 – Procedure TfrmPadraoDB.TipoManutencao

No formulário Menu ocorre as chamadas de criação dos formulários de cadastro de perguntas e níveis (Listagem 4).

```
procedure TfrmMenu.Perguntas1Click(Sender: TObject);
begin
 if not Assigned(frmPerguntas) then
 frmPerguntas := TfrmPerguntas.Create(nil)
 else
 msgOk('Este formulario já foi criado');
end;

procedure TfrmMenu.Nveis1Click(Sender: TObject);
```

```
begin
 if not Assigned(frmNiveis) then
 frmNiveis := TfrmNiveis.Create(nil)
 else
 msgOk('Este formulario já foi criado');
end;
```

Listagem 4 – Procedure TfrmMenu.Perguntas1Click e Procedure TfrmMenu.Nveis1Click

Ainda no formulário Menu ocorrem as chamadas de criação dos formulários de cadastro de categorias e a opção sair. O código da *procedure* TfrmMenu.Sair1Click e Procedure TfrmMenu.Categorias1Click está na Listagem 5.

```
procedure TfrmMenu.Sair1Click(Sender: TObject);
begin
 close;
end;

procedure TfrmMenu.Categorias1Click(Sender: TObject);
begin
 if not Assigned(frmCategoria) then
 frmCategoria := TfrmCategoria.Create(nil)
 else
 msgOk('Este formulario já foi criado');
end;
```

Listagem 5 – Procedure TfrmMenu.Sair1Click e Procedure TfrmMenu.Categorias1Click

Para cada formulário (níveis, categorias e perguntas) foi implementada uma função para validar os dados chamada de “ValidaDados”. Como pode ser observado na Listagem 6 são validados os dados do “edtCodNivel” solicitando obrigatoriamente um código e o “edtDescricao” solicitando obrigatoriamente uma descrição para o nível da pergunta.

```
function TfrmNiveis.ValidaDados: boolean;
begin
 Result := true;
 if (edtCodNivel.Text = '') then
 begin
 msgOK('Informe o Cód. do Nível');
 edtCodNivel.SetFocus;
 Result := false;
 end
 else if (edtDescricao.Text = '') then
 begin
 msgOK('Informe a Descrição do Nível');
 edtDescricao.SetFocus;
 Result := false;
 end
end;
```

Listagem 6 – Function TfrmNiveis.ValidaDados

Como pode ser observado na Listagem 7 são validados os dados do “edtCodCateg” solicitando obrigatoriamente um código e o “edtDescricao” e uma descrição para a categoria da pergunta.

```
function TfrmCategoria.ValidaDados: boolean;
begin
 Result := true;
 if (edtCodCateg.Text = '') then
 begin
 msgOK('Informe o Cód. da Categoria');
 edtCodCateg.SetFocus;
 Result := false;
 end
 else if (edtDescricao.Text = '') then
 begin
 msgOK('Informe a Descrição da Categoria');
 edtDescricao.SetFocus;
 Result := false;
 end
end;
```

Listagem 7 – Function TfrmCategoria.ValidaDados

A Listagem 8 apresenta como são validados os dados do “edtCodPergunta” solicitando um código, o “edtPergunta” solicitando uma pergunta, “edtAltCorreta” solicitando a resposta correta para a pergunta, “edtAltern” solicitando uma outra alternativa de resposta para a pergunta. “lcbNivel” e “lcbCat” são opções a partir de dados cadastrados no banco e enviados diretamente ao formulário para a escolha do nível da pergunta e da categoria da pergunta.

```
function TfrmPerguntas.ValidaDados: boolean;
begin
Result := true;
 if (edtCodPergunta.Text = '') then
 begin
 msgOK('Informe o Cód. da Pergunta');
 edtCodPergunta.SetFocus;
 Result := false;
 end
 else if (edtPergunta.Text = '') then
 begin
 msgOK('Informe a Pergunta');
 edtPergunta.SetFocus;
 Result := false;
 end
 else if (edtAltCorreta.Text = '') then
 begin
 msgOK('Informe a alternativa correta da Pergunta');
 edtAltCorreta.SetFocus;
 Result := false;
 end
 else if (edtAltern1.Text = '') then
 begin
 msgOK('Informe uma outra alternativa para a Pergunta');
```

```
 edtAltern1.SetFocus;
 Result := false;
 end
 else if (lcbNivel.Text = '') then
 begin
 msgOK('Informe o nivel da Pergunta');
 lcbNivel.SetFocus;
 Result := false;
 end
 else if (lcbCat.Text = '') then
 begin
 msgOK('Informe a Categoria da Pergunta');
 lcbCat.SetFocus;
 Result := false;
 end;

end;
```

Listagem 8 – Function TfrmPerguntas.ValidaDados

4.4.2 Módulo Responder Perguntas

Os comandos da *procedure TfrmResponder.FormClose* conforme apresentado na Listagem 9, ocorrem no fechamento do formulário. Nesta *procedure* verifica-se a existência de mais perguntas para o teste, ou seja, é verificado se o teste foi totalmente respondido. Caso esteja totalmente respondido é passado o número de respostas armazenadas no cdsGravar e apresenta mensagem. Caso o teste não esteja completo o mesmo não é gravado.

```
procedure TfrmResponder.FormClose(Sender: TObject;
 var Action: TCloseAction);
begin
 if cds.Eof then
 begin
 try
 if cdsGravar.ApplyUpdates(0) = 0 then
 msgOK('Teste salvo com sucesso!')
 else
 msgOK('Erro ao gravar o registro!');
 except
 msgOK('Problemas ao gravar o registro!');
 end;
 end
 else
 ShowMessage('O teste está incompleto e não será salvo!');
 cdsGravar.Close;
 cds.Close;
 Action := caFree;
 frmResponder := nil;
end;
```

Listagem 9 - Procedure TfrmResponder.FormClose

Na "*procedure TfrmResponder.FormShow"* da Listagem 10, no evento onShow do formulário ocorre a consulta de quantos testes o usuário logado já fez. Se é o primeiro teste realizado é chamada a *procedure* PrimeiroTeste. Se o usuário já realizou algum teste anterior é chamada a *procedure* SegundoTeste. Neste evento é verificado quantas categorias o teste possui e quantas perguntas serão atribuídas para cada categoria. Em seguida é chamada a *procedure* AtualizaPergunta.

```
procedure TfrmResponder.FormShow(Sender: TObject);
begin
 numAcessos := 0;
 totalPerguntas := 0;
 usuario := frmMenu.usuarioLogado;
 qryTeste.Close;
 qryTeste.SQL.Text := 'SELECT COUNT(idteste) as NUM_ACESSOS FROM TESTE '
+'WHERE teste.usuario = ' + quotedStr(usuario);
 qryTeste.Open;
 numAcessos := qryTeste.FieldByName('NUM_ACESSOS').AsInteger;
 qryTeste.Close;
 qryTeste.SQL.Text := 'SELECT COUNT(DISTINCT PERGUNTAS.CODCATEGORIA) AS
QUANTIDADE FROM PERGUNTAS ' ;
 qryTeste.Open;
 numCategorias := qryTeste.FieldByName('QUANTIDADE').AsInteger;
 if numAcessos > 0 then
 SegundoTeste
 Else
 PrimeiroTeste;
 qryTeste.close;
 qryTeste.SQL.Text := 'SELECT MAX(IDTESTE) + 1 AS PROXIMO FROM TESTE';
 qryTeste.Open;
 nIDTeste := qryTeste.FieldByName('PROXIMO').AsInteger;
 qryTeste.close;
 CDS.First;
 cdsGravar.Open;
 AtualizaPergunta;
end;
```

Listagem 10 - Procedure TfrmResponder.FormShow

Na *procedure TfrmResponder.AtualizaPergunta*, Listagem 11, verifica-se se a pergunta selecionada no banco possui três ou quatro alternativas para atribuição nos RadioButtons. É atribuída a pergunta ao MemPergunta e as alternativas aos RadioButtons de forma aleatória atribuindo à Tag o valor 1 se aquela alternativa for a correta e 0 se a alternativa for a incorreta. Esta atribuição de valores à s Tags é utilizada para a correção.

```
procedure TfrmResponder.AtualizaPergunta;
var
 nSorteio : smallint;
begin
 btnConferir.Enabled := True;
 pnlAlternativas.Enabled := True;
 lblresult.Caption := '';
 rdbAltern1.Checked := False;
```

```
 rdbAltern2.Checked := False;
 rdbAltern3.Checked := False;
 rdbAltern4.Checked := False;
 MemPergunta.Lines.Text := IntToStr(totalPerguntas + 1) + ' - ' +
 cds.FieldByName('PERGUNTA').AsString;

 if cds.FieldByName('ALTERNATIVA3').AsString <> '' then
 begin
 nSorteio := Random(4) + 1;
 rdbAltern4.Visible := True;
 end
 else
 begin
 nSorteio := Random(3) + 1;
 rdbAltern4.Visible := False;
 end;
 rdbAltern1.Tag := 0;
 rdbAltern2.Tag := 0;
 rdbAltern3.Tag := 0;
 rdbAltern4.Tag := 0;
 case nSorteio of
 1: begin
 rdbAltern1.Caption:= cds.FieldByName('ALTERNATIVACORRETA').AsString;
 rdbAltern1.Tag := 1;
 rdbAltern2.Caption := cds.FieldByName('ALTERNATIVA1').AsString;
 rdbAltern3.Caption := cds.FieldByName('ALTERNATIVA2').AsString;
 rdbAltern4.Caption := cds.FieldByName('ALTERNATIVA3').AsString;
 end;
 2: begin
 rdbAltern1.Caption := cds.FieldByName('ALTERNATIVA1').AsString;
 rdbAltern2.Caption:= cds.FieldByName('ALTERNATIVACORRETA').AsString;
 rdbAltern2.Tag := 1;
 rdbAltern3.Caption := cds.FieldByName('ALTERNATIVA2').AsString;
 rdbAltern4.Caption := cds.FieldByName('ALTERNATIVA3').AsString;
 end;
 3: begin
 rdbAltern1.Caption := cds.FieldByName('ALTERNATIVA1').AsString;
 rdbAltern2.Caption := cds.FieldByName('ALTERNATIVA2').AsString;
 rdbAltern3.Caption:= cds.FieldByName('ALTERNATIVACORRETA').AsString;
 rdbAltern3.Tag := 1;
 rdbAltern4.Caption := cds.FieldByName('ALTERNATIVA3').AsString;
 end;
 4: begin
 rdbAltern1.Caption := cds.FieldByName('ALTERNATIVA1').AsString;
 rdbAltern2.Caption := cds.FieldByName('ALTERNATIVA2').AsString;
 rdbAltern3.Caption := cds.FieldByName('ALTERNATIVA3').AsString;
 rdbAltern4.Caption:= cds.FieldByName('ALTERNATIVACORRETA').AsString;
 rdbAltern4.Tag := 1;
 end;
 end;
 totalPerguntas := totalPerguntas +1;
end;
```

Listagem 11 - procedure TfrmResponder.AtualizaPergunta

Quando acionado o botão Conferir é executada a *procedure* TfrmResponder.btnconferirClick, Listagem 12, que verifica se foi marcada a resposta certa ou qualquer outra resposta. Caso tenha sido marcada a resposta certa a *procedure* GravarTeste é

executada tendo como parâmetro o valor 1. Caso tenha sido marcada qualquer outra resposta a *procedure* GravarTeste é executada tendo como parâmetro o valor 0. O *label* lblResult recebe o texto 'Resposta Correta' em verde ou "Resposta Incorreta" em vermelho. Caso não seja selecionada nenhuma alternativa é apresentada a mensagem "A questão não foi respondida".

```
procedure TfrmResponder.btnconferirClick(Sender: TObject);
begin
 if ((rdbAltern1.Checked) and (rdbAltern1.Tag = 1)) or
 ((rdbAltern2.Checked) and (rdbAltern2.Tag = 1)) or
 ((rdbAltern3.Checked) and (rdbAltern3.Tag = 1)) or
 ((rdbAltern4.Checked) and (rdbAltern4.Tag = 1)) then
 begin
 btnproximo.Enabled := True;
 btnConferir.Enabled := False;
 pnlAlternativas.Enabled := False;
 lblresult.Caption := 'Resposta correta';
 lblresult.Font.Size := 16;
 lblresult.Font.Color := clGreen;
 GravarTeste(1);
 end
 else if ((rdbAltern1.Checked) or
 (rdbAltern2.Checked) or
 (rdbAltern3.Checked) or
 (rdbAltern4.Checked)) then
 begin
 btnproximo.Enabled := True;
 btnConferir.Enabled := False;
 pnlAlternativas.Enabled := False;
 lblresult.Caption := 'Resposta incorreta';
 lblresult.Font.Size := 16;
 lblresult.Font.Color := clred;
 GravarTeste(0);
 end
 else
 msgOK('A questão não foi respondida!');
end;
```

Listagem 12 - procedure TfrmResponder.btnconferirClick

Quando acionado o botão Próximo executa a *procedure* TfrmResponder.btnproximoClick, apresentada Listagem 13, que verifica se não é o fim do cds (componente ClientDataSet) a fim de identificar se ainda há perguntas. Chama a procedure AtualizaPergunta e deixa desabilitado o botão "Próximo". Se for o fim do cds apresenta mensagem de questionário encerrado e que o resultado está disponível para consulta na opção do menu Relatórios.

```
procedure TfrmResponder.btnproximoClick(Sender: TObject);
begin
 cds.Next;
 if not cds.Eof then
 begin
 AtualizaPergunta;
 btnproximo.Enabled := False;
```

```
 end
 else
 begin
 lblresult.Caption := '';
 btnproximo.Enabled := False;
 msgOK('Questionário Encerrado!'#13#13 +
 'O resultado já está disponível em'#13'Relatórios > Ranking');
 Close;
 end;
end;
```

Listagem 13 - procedure TfrmResponder.btnproximoClick

A "*procedure TfrmResponder.GravarTeste*", Listagem 14, é responsável por armazenar o resultado de cada questão do teste atribuindo ao cdsGravar (componente ClientDataSet).

```
procedure TfrmResponder.GravarTeste(nAcerto : smallint);
begin
 cdsGravar.Append;
 cdsGravar.FieldByName('IDTESTE').AsInteger := nIDTeste;
 cdsGravar.FieldByName('USUARIO').AsString :=
 frmMenu.StatusBar1.Panels[1].Text;
 cdsGravar.FieldByName('CODPERGUNTA').AsInteger :=
 cds.FieldByName('CODPERGUNTA').AsInteger;
 cdsGravar.FieldByName('CORRETA').AsInteger := nAcerto;
 cdsGravar.FieldByName('ERRADA').AsInteger := 1 - nAcerto;
 cdsGravar.Post;
end;
```

Listagem 14 - procedure TfrmResponder.GravarTeste

Na *procedure TfrmResponder.PrimeiroTeste* da Listagem 15, o teste é composto por uma quantidade igual de perguntas para cada categoria. Para determinar a quantidade de perguntas de cada categoria que compõe o teste, faz-se uma busca no banco de dados a fim de contar a quantidade de categorias utilizadas em perguntas ("select count distinct"). É dividido o total de perguntas do teste (20) pela quantidade de categorias para definir quantas perguntas serão selecionadas de cada categoria. Para composição do teste, as perguntas para cada categoria são sorteadas randomicamente do banco de dados. Posteriormente é atribuído ao cds a pergunta e as respectivas alternativas.

```
procedure TfrmResponder.PrimeiroTeste;
var
 numPerguntas : Double;
 codCategoria, filtraPerguntas: String;
begin
 qryTeste.Close;
 qryTeste.SQL.Text := 'SELECT COUNT(DISTINCT PERGUNTAS.CODCATEGORIA) AS
QUANTIDADE ' +
 ' FROM PERGUNTAS ' ;
 qryTeste.Open;
 numPerguntas := (20/qryTeste.FieldByName('QUANTIDADE').AsInteger);
```

```
 qryTeste.Close;
 qryTeste.SQL.Text := 'SELECT FIRST 20 DISTINCT codcategoria ' +
 ' from PERGUNTAS ORDER BY RAND() ' ;
 qryTeste.Open;
 while not qryTeste.eof do
 begin
 CodCategoria := qryTeste.FieldByName('CODCATEGORIA').AsString;
 qryTeste.Next;
 if not qryTeste.Eof then
 filtraPerguntas := 'first '+ IntToStr(Trunc(numPerguntas))
 else
 filtraPerguntas := '';

 qryresponder.Close;
 qryresponder.SQL.Text := 'SELECT '+ filtraPerguntas + ' * FROM perguntas '
+
 ' WHERE PERGUNTAS.CODCATEGORIA = ' + codCategoria +
 ' ORDER BY RAND()';
 qryresponder.open;
 while not (qryresponder.eof) and (cds.RecordCount < 20) do
 begin
 CDS.Append;
 CDS.fieldByName('CODPERGUNTA').AsInteger :=
qryresponder.fieldByName('CODPERGUNTA').AsInteger;
 CDS.fieldByName('PERGUNTA').AsString :=
qryresponder.fieldByName('PERGUNTA').AsString;
 CDS.fieldByName('ALTERNATIVACORRETA').AsString :=
qryresponder.fieldByName('ALTERNATIVACORRETA').AsString;
 CDS.fieldByName('ALTERNATIVA1').AsString :=
qryresponder.fieldByName('ALTERNATIVA1').AsString;
 CDS.fieldByName('ALTERNATIVA2').AsString :=
qryresponder.fieldByName('ALTERNATIVA2').AsString;
 CDS.fieldByName('ALTERNATIVA3').AsString :=
qryresponder.fieldByName('ALTERNATIVA3').AsString;
 CDS.fieldByName('CODNIVEL').AsInteger :=
qryresponder.fieldByName('CODNIVEL').AsInteger;
 CDS.fieldByName('CODCATEGORIA').AsInteger :=
qryresponder.fieldByName('CODCATEGORIA').AsInteger;
 CDS.Post;
 qryresponder.Next;
 end;
 end;
 qryresponder.Close;
end;
```

Listagem 15 - procedure TfrmResponder.PrimeiroTeste

Na *procedure TfrmResponder.SegundoTeste*, Listagem 16, ordena a quantidade de erros do teste anterior do usuário pelas categorias das perguntas. As três categorias que tiverem maior quantidade de erros recebem três perguntas cada (considera-se que o teste é composto por 20 perguntas). As demais perguntas que completam o teste são selecionadas em igual número das demais categorias, excluindo as que já receberam as perguntas em virtude dos erros do teste anterior. Caso ao final ainda faltem perguntas para preencher o total de

perguntas do teste é selecionado mais perguntas da última categoria verificada, sem repetir as perguntas já selecionadas.

```
procedure TfrmResponder.SegundoTeste;
var
 numPerguntas : Double;
 CategoriasUsadas, FiltraPerguntas, CodCategoria : String;
 CategoriasComErro : smallint;

begin
qryTeste.Close;
 qryTeste.SQL.Text :=
 'SELECT FIRST 3 teste.idteste, PERGUNTAS.codcategoria, COUNT(TESTE.errada)
AS ERROS ' +
 ' FROM TESTE '+
 ' LEFT JOIN PERGUNTAS '+
 ' ON (PERGUNTAS.codpergunta = teste.codpergunta) '+
 ' WHERE (TESTE.usuario = '+QuotedStr(usuario)+') AND '+
 ' (TESTE.ERRADA = 1) AND '+
 ' (TESTE.idteste = '+
 ' (SELECT MAX(T.IDTESTE) '+
 ' FROM TESTE AS T '+
 ' WHERE (T.USUARIO = TESTE.USUARIO))) '+
 ' GROUP BY teste.idteste, PERGUNTAS.codcategoria '+
 ' ORDER BY COUNT(TESTE.ERRADA) DESC ';
 qryTeste.Open;
 CategoriasUsadas := '0';
 CategoriasComErro := 0;
 while not qryteste.Eof do
 begin
 CategoriasUsadas := CategoriasUsadas + ', '
+qryTeste.FieldByName('CODCATEGORIA').AsString;
 Inc(CategoriasComErro);
 qryresponder.Close;
 qryresponder.SQL.Text := 'SELECT first 3 * FROM perguntas ' +
 ' WHERE PERGUNTAS.CODCATEGORIA = ' +
qryTeste.FieldByName('CODCATEGORIA').AsString +
 ' ORDER BY RAND()';
 qryresponder.open;
 while not qryresponder.eof do
 begin
 CDS.Append;
 CDS.fieldByName('CODPERGUNTA').AsInteger :=
qryresponder.fieldByName('CODPERGUNTA').AsInteger;
 CDS.fieldByName('PERGUNTA').AsString :=
qryresponder.fieldByName('PERGUNTA').AsString;
 CDS.fieldByName('ALTERNATIVACORRETA').AsString :=
qryresponder.fieldByName('ALTERNATIVACORRETA').AsString;
 CDS.fieldByName('ALTERNATIVA1').AsString :=
qryresponder.fieldByName('ALTERNATIVA1').AsString;
 CDS.fieldByName('ALTERNATIVA2').AsString :=
qryresponder.fieldByName('ALTERNATIVA2').AsString;
 CDS.fieldByName('ALTERNATIVA3').AsString :=
qryresponder.fieldByName('ALTERNATIVA3').AsString;
 CDS.fieldByName('CODNIVEL').AsInteger :=
qryresponder.fieldByName('CODNIVEL').AsInteger;
 CDS.fieldByName('CODCATEGORIA').AsInteger :=
qryresponder.fieldByName('CODCATEGORIA').AsInteger;
 CDS.Post;
```

```
 qryresponder.Next;
 end;
 qryTeste.Next;
 end;

 numPerguntas := (20 - (3*CategoriasComErro))/(numCategorias -
CategoriasComErro);
 qryTeste.Close;
 qryTeste.SQL.Text := 'SELECT FIRST '+IntToStr(numCategorias -
CategoriasComErro)+' DISTINCT codcategoria ' +
 ' from PERGUNTAS ' +
 ' WHERE (NOT (CODCATEGORIA IN ('+CategoriasUsadas+'))) ORDER BY RAND() ' ;
 qryTeste.Open;
 while not qryTeste.eof do
 begin
 CodCategoria := qryTeste.FieldByName('CODCATEGORIA').AsString;
 qryTeste.Next;
 if not qryTeste.Eof then
 filtraPerguntas := 'first '+ IntToStr(Trunc(numPerguntas))
 else
 filtraPerguntas := '';
 qryresponder.Close;
 qryresponder.SQL.Text := 'SELECT '+filtraPerguntas+' * FROM perguntas ' +
 ' WHERE PERGUNTAS.CODCATEGORIA = ' + CodCategoria +
 ' ORDER BY RAND()';
 qryresponder.open;
 while not (qryresponder.eof) and (cds.RecordCount < 20) do
 begin
 CDS.Append;
 CDS.fieldByName('CODPERGUNTA').AsInteger :=
qryresponder.fieldByName('CODPERGUNTA').AsInteger;
 CDS.fieldByName('PERGUNTA').AsString :=
qryresponder.fieldByName('PERGUNTA').AsString;
 CDS.fieldByName('ALTERNATIVACORRETA').AsString :=
qryresponder.fieldByName('ALTERNATIVACORRETA').AsString;
 CDS.fieldByName('ALTERNATIVA1').AsString :=
qryresponder.fieldByName('ALTERNATIVA1').AsString;
 CDS.fieldByName('ALTERNATIVA2').AsString :=
qryresponder.fieldByName('ALTERNATIVA2').AsString;
 CDS.fieldByName('ALTERNATIVA3').AsString :=
qryresponder.fieldByName('ALTERNATIVA3').AsString;
 CDS.fieldByName('CODNIVEL').AsInteger :=
qryresponder.fieldByName('CODNIVEL').AsInteger;
 CDS.fieldByName('CODCATEGORIA').AsInteger :=
qryresponder.fieldByName('CODCATEGORIA').AsInteger;
 CDS.Post;
 qryresponder.Next;
 end;
 end;
 qryresponder.Close;
end;
```

Listagem 16 - procedure TfrmResponder.SegundoTeste

5 CONCLUSÃO

O desenvolvimento deste trabalho objetivou a criação de um software para cadastro de questões de múltipla escolha em um banco de dados. Esse banco foi utilizado para disponibilizar um módulo (programa) que permitisse responder testes compostos a partir do cadastro das perguntas.

A implementação foi realizada com a linguagem Delphi que permite a programação a partir de componentes e baseada em eventos. Assim, o referencial teórico abordou conceitos, vantagens e desvantagens da programação *desktop*, por eventos e com desenvolvimento a partir de componentes. Percebeu-se que apesar da grande ênfase atual no desenvolvimento para ambiente *web*, a programação *desktop* ainda tem muito espaço, seja pelos sistemas legados existentes ou mesmo pelas possibilidades que essas tecnologias oferecem, inclusive de vínculo com linguagens para *web*. Como ocorre, por exemplo, com Delphi e PHP.

A ferramenta Visual Paradigm foi utilizada para desenvolver a modelagem do sistema. Essa ferramenta foi de grande ajuda para fazer a análise do sistema, auxiliando a visualizar de forma gráfica as entidades de implementação (classe) e do banco de dados (tabelas). A linguagem Delphi 7 facilitou a implementação do sistema, inclusive, pela quantidade de componentes prontos que a mesma oferece.

Por fim, foi possível verificar que os objetivos do trabalho foram alcançados. O sistema implementado pode ser utilizado para cadastro de qualquer tipo de questões de múltipla escolha, como, por exemplo, provas do DETRAN e do ENEM. As perguntas são organizadas em categorias e níveis de dificuldade. Ao final da resolução do teste o usuário será informado da quantidade de erros e acertos. Foi implementada uma forma de compor o teste com base na quantidade de erros por categoria do teste anterior do usuário. É uma maneira de auxiliar o usuário nos conteúdos que ele apresenta maior dificuldade.

REFERÊNCIAS

BLOIS, A. P. T. B., **Uma abordagem de projeto arquitetural baseado em componentes no contexto de engenharia de domínio**. Tese (doutorado). COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2006. Disponível em: <http://www.reuse.cos.ufrj.br>. Acesso em: 20 mar. 2013

CAMPOS, Alexandre Soares. **Desenvolvimento desktop ou browser?**. Disponível em: <http://www.profissionaisdetecnologia.com.br/blog/?p=139>. Acesso em: 15 mar. 2013.

CANTU, Marco. **Dominando o Delphi 7 "A Bíblia"**. São Paulo: Makron Books, 2003.

D'SOUZA, Desmond Francis; WILLS, Alan Cameron. **Objects, components and frameworks with UML: the Catalysis Approach**. Reading: Addison-Wesley, 1998

FIREBIRD. **Firebird**. Disponível em: <http://www.firebirdsql.org/>. Acesso em: 20 nov. 2012.

IBEXPERT. **IBExpert developer sudio**. Disponível em: <http://www.ibexpert.com/>. Acesso em: 14 nov. 2012.

MACORATTI, José Carlos. **Desenvolvendo para desktop ou para web?** Disponível em: <http://www.macoratti.net/vbn_dkwb.htm>. Acesso em 24 de fev. 2013.

MARQUES, Paulo; PEDROSO, Hernâni, FIGUEIRA, Ricardo. **C# 3.5. Programação baseada em componentes**. FCA Editora de Informática. P. 141- 150. Disponível em: <http://www.fca.pt/docs-online/722-403-6_pags_141_150.pdf>. Acesso em: 25 mar. 2013.

MOURA, Ana M. M.; CARVALHO, Jonnathan dos S.; SILVA, Rogério de J. **Abordagens de reutilização de software e sua aplicação no domínio de arrecadação tributária municipal**. Monografia de Pós Graduação - CEFET (Centro Federal de Educação Tecnológica) Campos. 2006.

PRESSMAN, Roger. **Engenharia de software**. Rio de Janeiro: McGraw Hill 2006.

SILVEIRA, Marcelo P., VARGAS, Pablo Tôndolo de. **Paradigmas orientado a eventos**. 2007. Disponível: < http://www.dcc.unimontes.br/renato/2009/SDI/MATERIAIS/2007_1_paradigmas_orientado_eventos.pdf >. Acesso em: 17 mar. 2013.

SPAGNOLI, Luciana de Araújo; BECKER, Karin. **Um estudo sobre o desenvolvimento baseado em componentes**. Faculdade de Informática. Technical Report Series, n. 26, Maio de 2003.

FAST REPORT. **Fast Report**. Disponível em: http://www.fast-report.com/public_download/FR4.6.UserManual-en.chm. Acesso em: 21/04/2013

VISUAL PARADIGM. **Visual Paradigm**. Disponível em: <http://www.visual-paradigm.com/>. Acesso em: 23 out. 2012.