UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE COMPUTAÇÃO E SISTEMAS

ÍTALO ALMEIDA AGUIAR

ARCHIDATA: FERRAMENTA PARA MODELAGEM DE BANCO DE DADOS COM SUPORTE A EXTENSÕES E MÚLTIPLOS SGBDS

João Monlevade
2017

ÍTALO ALMEIDA AGUIAR

ARCHIDATA: FERRAMENTA PARA MODELAGEM DE BANCO DE DADOS COM SUPORTE A EXTENSÕES E MÚLTIPLOS SGBDS

Monografia apresentada ao curso de Sistemas de Informação do Instituto de Ciências Exatas e Aplicadas, da Universidade Federal de Ouro Preto, como requisito parcial para aprovação na Disciplina “Trabalho de Conclusão de Curso II”.

Orientador: Eduardo da Silva Ribeiro

Co-Orientador: Euler Horta Marinho

João Monlevade

2017

UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E APLICADAS
COLEGIADO DO CURSO DE SISTEMAS DE INFORMAÇÃO

ATA DE DEFESA

Aos 30 dias do mês de Março de 2017, às 19 horas, na sala C204 do Instituto de Ciências Exatas e Aplicadas, foi realizada a defesa de Monografia pelo aluno Ítalo Almeida Aguiar, sendo a Comissão Examinadora constituída pelos professores: Prof. Me. Eduardo da Silva Ribeiro, Prof. Me. Euler Horta Marinho e Prof. Me. Bruno Rabello Monteiro.

O candidato apresentou a monografia intitulada: "Desenvolvimento de Ferramenta Gráfica Extensível para Modelagem de Banco de Dados". A comissão examinadora deliberou, por unanimidade, pela aprovação do candidato, com nota 9,5 (nove e meio), concedendo-lhe o prazo de 15 dias para incorporação das alterações sugeridas ao texto final.

Na forma regulamentar, foi lavrada a presente ata que é assinada pelos membros da Comissão Examinadora e pelo graduando.

João Monlevade, 30 de Março de 2017.

Prof. Me. Eduardo da Silva Ribeiro
Professor Orientador/Presidente

Prof. Me. Euler Horta Marinho
Professor Convidado

Prof. Me. Bruno Rabello Monteiro
Professor Convidado

Ítalo Almeida Aguiar
Graduando

ANEXO III – Termo de Responsabilidade

TERMO DE RESPONSABILIDADE

Eu, ÍTALO ALMEIDA AGUIAR, declaro que o texto do trabalho de conclusão de curso intitulado "**ARCHIDATA: FERRAMENTA PARA MODELAGEM DE BANCO DE DADOS COM SUPORTE A EXTENSÕES E MÚLTIPLOS SGBDS**" é de minha inteira responsabilidade e que não há utilização de texto, material fotográfico, código fonte de programa ou qualquer outro material pertencente a terceiros sem as devidas referências ou consentimento dos respectivos autores.

João Monlevade, 30 de março de 2017

Ítalo Almeida Aguiar
Assinatura do aluno

DEDICATÓRIA

Dedico este trabalho primeiramente a Deus que sempre guiou meu caminho por mais que a estrada fosse tortuosa e a minha família que sempre se fez presente nos momentos que mais precisei.

RESUMO

O uso de ferramentas de *software* para a modelagem de bancos de dados permite a desenvolvedores, administradores e arquitetos de bancos de dados projetar a estrutura de armazenamento como uso de notações gráficas. Este trabalho propõe o projeto e desenvolvimento de um *software* de modelagem de banco de dados, que permita através de notações gráficas, modelar e produzir todo o código de criação da respectiva estrutura de banco de dados. Este trabalho também aborda os principais testes de software pelos quais a ferramenta proposta foi submetida como forma de validação e verificação, resultando no aumento da qualidade do produto final.

Palavras-chave: Banco de dados. Modelagem. *Software*. Desenvolvimento. Testes.

ABSTRACT

The use of software tools for database modeling allows developers, administrators, and database architects to design the graphical-based storage structure. This work proposes the design and development of a database modeling tool that allows the use of graphical notations to model and produce the creation code of the respective database structure. This work also presents the software tests used for validation and verification, resulting in the increase of the quality of the final product.

Keywords: Database. Modeling. Software. Development. Tests

LISTA DE FIGURAS

Figura 1: Componentes do .NET.....19
Figura 2 Exemplo de elemento estendido via XAML (FlyoutMenu) 21
Figura 3: Estrutura do padrão MVVM.....22
Figura 4: Exemplo de uso da interface *INotifyPropertyChanged* 23
Figura 5: Estrutura do *software* proposto 24
Figura 6: Estrutura MEF para o *plug-in* ER 25
Figura 7: Tela de Carregamento para o *software* proposto 26
Figura 8: Interface gráfica final da tela principal.....27
Figura 9: Conjunto de ferramentas expostas pelo *plug-in* 28
Figura 10: Exemplo de Diagrama criado no *plug-in* ER.....28
Figura 11: Editor de propriedades do *plug-in* 29
Figura 12: *Plug-in* editor de código SQL 30
Figura 13: Ícone da aplicação.....31
Figura 14: Exemplo de classe de teste de unidade.....33
Figura 15: Exemplo de resultado de teste reportado pelo ambiente de desenvolvimento.....34
Figura 16: Caso de uso do *software* 42
Figura 17: Diagrama ER para reprodução em testes 45

LISTA DE TABELAS

Tabela 1: Exceções lançadas durante o teste de *software* 36
Tabela 2: Problemas relatados pelos usuários.....37
Tabela 3: Trabalhos futuros observados para o *software*.....38

LISTA DE ABREVIATURAS

SGBD – Sistema de Gerenciamento de Bancos de Dados

ER – Entidade-Relacionamento (Modelo)

MEF – *Managed Extensibility Framework* (*Framework* de Extensibilidade Gerenciado)

UML – *Unified Modeling Language* (Linguagem Unificada de Modelagem)

LINQ – Language Integrated Query (Linguagem de Consulta Integrada)

GC – Garbage Collector (Coletor de Lixo)

CPU – Unidade Central de Processamento

MSIL – Microsoft *Intermediate Language* (Linguagem Intermediária da Microsoft)

WPF – *Windows Presentation Foundation*

XAML – *Extensible Application Markup Language* (Linguagem de Marcação de Aplicação Extensível)

XML – *Extensible Markup Language* (Linguagem de Marcação Extensível)

HTML – *HyperText Markup Language* (Linguagem de Marcação de Hipertexto)

DLL – *Dynamic-link library* (Biblioteca de vínculo dinâmico)

SUMÁRIO

1 INTRODUÇÃO 14

1.1 PROBLEMA 14

1.2 OBJETIVOS 15

1.2.1 Objetivo geral 15

1.2.2 Objetivos específicos 15

1.3 JUSTIFICATIVA 15

2 CONCEITOS GERAIS E REVISÃO DA LITERATURA 16

3 PROJETO DE *SOFTWARE* 17

3.1 ANÁLISE DE REQUISITOS 17

3.2 TECNOLOGIAS ADOTADAS 18

3.2.1 Plataforma de desenvolvimento .NET *Framework* 18

3.2.2 Linguagem de Programação C# (C *Sharp*) 20

3.2.3 *Windows Presentation Foundation* (WPF) 20

3.3 PADRÕES DE PROJETO 21

3.3.1 MVVM 21

3.3.2 *Memento* 22

3.3.3 *Observer* 23

3.4 ESTRUTURA E ARQUITETURA DO *SOFTWARE* 23

3.5 *DESIGN* DE INTERFACE GRÁFICA 25

3.5.1 *Splash Screen* (Tela de Carregamento) 26

3.5.2 Tela Principal 26

3.5.3 *Plug-in* Diagrama Entidade-Relacionamento 28

3.5.4 *Plug-in* editor de código SQL 29

4 ARCHIDATA 31

5 TESTE DE *SOFTWARE* 32

5.1 TESTE DE UNIDADE 32

5.1.1 Executando testes de unidade no *software* 33
5.2 TESTE *ALFA* 34
5.2.1 Ambiente de Testes..... 35
5.2.2 Condução dos Testes..... 35
5.2.3 RESULTADOS E DISCUSSÕES DO TESTE ALFA 36
6 TRABALHOS FUTUROS..... 38
7 CONCLUSÕES 39
REFERÊNCIAS..... 40
APÊNDICE A 42
APÊNDICE B 43
APÊNDICE C 44
APÊNDICE D 45
APÊNDICE E 46
APÊNDICE F..... 49
APÊNDICE G 51
APÊNDICE H 53

1 Introdução

Atualmente os sistemas computacionais estão cada vez mais presentes no cotidiano das pessoas. Sempre que um novo sistema é idealizado, surge a necessidade da realização de um levantamento de todos os requisitos necessários para que a aplicação possa ser desenvolvida. A maioria dos sistemas necessita que pelo menos uma parte de seus dados sejam persistidos de alguma forma (ELMASRI e NAVATHE, 2011).

Para auxiliar o trabalho de gerenciar os dados das aplicações, foram desenvolvidos os chamados Sistemas de Gerenciamento de Banco de Dados, ou SGBDs. Esses sistemas atuam de forma independente de outros sistemas, eliminando, desta forma, todo o trabalho de gestão e integridade dos dados a serem armazenados por uma aplicação (CHU, 1990).

Embora os SGBDs possam trazer grandes facilidades no desenvolvimento de novos sistemas, é necessário que a estrutura dos dados a serem armazenados seja previamente informada. Um projeto adequado desta estrutura de dados possibilitará ao SGBD, de certa forma, garantir que os dados estejam armazenados corretamente.

Para auxiliar a tarefa de projetar a estrutura do banco de dados, foram desenvolvidos diagramas visuais, entre os principais, temos o modelo Entidade-Relacionamento proposto por Peter Chen em 1976 (CHEN, 1976), e o modelo UML (*Unified Modeling Language*) (ELMASRI e NAVATHE, 2011).

1.1 Problema

Apesar dos diagramas de modelagem de bancos de dados permitirem que estes sejam escritos manualmente, foram desenvolvidos vários *softwares* para automatizar este processo de modelagem. Estes *softwares*, em sua maioria, apresentam interfaces gráficas onde é possível desenhar visualmente uma determinada notação de diagrama de modelagem de banco de dados, como por exemplo, o diagrama Entidade-Relacionamento (CHEN, 1976).

Estes sistemas, geralmente estão diretamente associados a um Sistema de SGBD específico, o que não possibilita que o trabalho realizado em um *software* possa ser aplicado a vários SGBDs distintos. Além disso, a maioria destes sistemas não permite expandir suas funcionalidades a outros SGBDs, ou trabalhar com mais de um tipo de diagrama de modelagem através de extensões. Ainda que alguns sistemas sejam mais completos, com uma grande variedade de funcionalidades, seu uso está atrelado a uma licença de *software*, que em sua grande maioria, não é gratuita.

1.2 Objetivos

Este trabalho procura alcançar como objetivo o desenvolvimento de um *software* para modelagem de bancos de dados. Além disso, um conjunto de objetivos secundários, que são considerados no desenvolvimento deste trabalho, são descritos nos objetivos específicos.

1.2.1 Objetivo geral

Desenvolver um *software* para a modelagem gráfica de bancos de dados, que inclua um conjunto mínimo de funcionalidades para a realização da tarefa de modelagem.

1.2.2 Objetivos específicos

- Prover através do *software* proposto um conjunto de ferramentas que possibilitem a modelagem gráfica de novos bancos de dados.
- Realizar um conjunto de testes sobre o *software* proposto de forma a permitir a descobertas de erros e sua respectiva correção de forma a aumentar a qualidade do produto final.
- Expor uma interface que possibilite a obtenção de um novo diagrama através de um banco de dados existente (engenharia reversa).
- Abstrair uma estrutura de *software* que permita estender de forma simplificada as funcionalidades da aplicação.

1.3 Justificativa

A maioria dos bancos de dados e SGBDs permitem a criação manual de novos bancos de dados. O uso de ferramentas automatizadas tem como principais objetivos, agilizar a etapa de criação e evitar que possíveis erros sejam transferidos ao banco de dados, quando modelado de forma manual. O uso de um modelo abstrato, como o modelo entidade-relacionamento, permite ao *designer* avaliar a criação de toda a estrutura em uma perspectiva alto-nível, reduzindo as chances de erro.

O desenvolvimento em código aberto *(open source)* da atual ferramenta permitirá que novos programadores contribuam com o projeto e amplie quantitativamente e qualitativamente as funcionalidades do *software*. O código aberto também permitirá que desenvolvedores e/ou empresas poupem custos com aquisição de licenças de ferramentas de modelagem.

2 Conceitos gerais e revisão da literatura

O uso de modelos gráficos para a criação de bancos de dados permitem arquitetar a forma com que os dados são armazenados em SGBDs de forma mais simples. As formalizações de modelos como o Entidade Relacionamento (CHEN, 1976) e UML (*Unified Modeling Language*) permitem criar *softwares* com o propósito de modelagem de banco de dados baseado nessas formalizações.

Entre os principais *softwares* de modelagem disponíveis no mercado temos: IBM *InfoSphere Data Architect*[1], erwin *Data Modeler*[2], *Power Design*[3]. Como observado, a grande maioria destes *softwares* são proprietários, ou seja, não há disponibilidade de código fonte para os usuários, já que o *software* é um artefato de propriedade intelectual da empresa.

Embora os *softwares* proprietários sejam repletos de funcionalidades, em determinado momento, um usuário pode se deparar com uma necessidade não prevista no *software*. Mesmo tendo pago um valor por uma licença do *software*, o usuário poderá não encontrar uma solução para seu problema com o *software*, já que o desenvolvedor do *software* dificilmente irá atender a uma solicitação de funcionalidade.

Como uma alternativa às soluções comerciais, o uso de *softwares* de código aberto, além de permitir seu uso gratuito, o próprio usuário poderá editar seu código caso deseje. O desenvolvimento de *software* livre permite que desenvolvedores que venham a ter contato com determinado *software* possam colaborar livremente para a melhoria do *software*, seja corrigindo uma falha, adicionando recursos ou melhorando características já existentes.

A extensibilidade de um *software* é também uma característica importante, pois, além de permitir que um *software* ganhe novas funcionalidades mais facilmente, também elimina a necessidade de se instalar novas versões do *software* sempre que uma nova funcionalidade é adicionada. Em alguns casos, um *software* pode ser estendido a partir de arquivos de notação textual, como por exemplo, XML (*Extensible Markup Language*).

Arquivos de texto podem permitir que até mesmo usuários menos experientes possam modificar um recurso do *software* se baseando em um já existente. Dependendo do nível de complexidade de uma funcionalidade estendida, esta poderá ser disponibilizada exclusivamente de forma compilada, ou seja, não é possível editar. A extensibilidade torna ainda mais simples a colaboração de usuários no desenvolvimento de um *software*, já que um usuário pode colaborar especificamente com determinado recurso.

---

[1] http://ibm.com/software/products/ibminfodataarch

[2] http://erwin.com/products/data-modeler

[3] http://powerdesigner.de

3 Projeto de *Software*

Para Pressman (PRESSMAN, 2011), a atividade de projeto de *software* engloba o conjunto de princípios, conceitos e práticas que levam ao desenvolvimento de um sistema ou produto com alta qualidade. A etapa de projeto é o momento de preparação para o desenvolvimento, onde todos os recursos, tecnologias, características e requisitos são reunidos para conduzir o desenvolvimento do *software* idealizado.

3.1 Análise de Requisitos

Uma aplicação destinada a executar um conjunto de ferramentas que possibilitem ao desenvolvedor, projetar e arquitetar as estruturas do banco de dados deve possuir um conjunto mínimo de características. Este conjunto mínimo de funcionalidades que deverá estar presente na aplicação é chamado de requisitos funcionais. Sommerville (SOMMERVILLE, 2007) especifica como requisitos funcionais, os requisitos que descrevem o que o sistema deve fazer. Conforme podemos observar nos casos de usos apresentados no APÊNDICE A, o software deve permitir ao usuário realizar um conjunto mínimo de ações durante sua utilização. A seguir podemos observar a lista com os requisitos funcionais levantados para o *software* proposto:

- Criação de Diagrama Entidade-Relacionamento (ER)
- Geração de código SQL a partir de um diagrama
- Salvar e abrir arquivos de projeto, tais como diagramas, códigos SQL, etc.
- Imprimir
- Exportar conteúdo como imagem e/ou outros formatos
- Obter um diagrama a partir de uma estrutura já existente (engenharia reversa)
- Ferramenta nativa de Zoom (se aplicável ao *plug-in* em uso)
- Copiar, recortar e colar dados (se aplicável ao *plug-in* em uso)

Os requisitos funcionais expressam características da aplicação que são tangíveis ao usuário, ou seja, podem ser invocadas ou manipuladas diretamente pelo usuário. Os requisitos não tangíveis ao usuário são chamados de **requisitos não funcionais**. Segundo Sommerville (SOMMERVILLE, 2007), os requisitos não funcionais são aqueles não diretamente relacionados às funções específicas fornecidas pelo sistema. Em outras palavras, os requisitos não funcionais expressam características as quais o usuário não consegue manipular diretamente, mas seu impacto poderá ser percebido durante a utilização do

software. A seguir, a lista com os requisitos não funcionais levantados para o desenvolvimento do *software* proposto:

- Facilidade de Uso
- Confiabilidade
- Suporte a *plug-ins* (Extensões de Software)

Ainda que a etapa de levantamento de requisitos seja feita de forma a prever todas as possíveis características que deverão estar presentes no *software*, novos requisitos poderão ser adicionados posteriormente. Em geral, a alteração de requisitos se dá em função de solicitação por parte do usuário, que por sua vez, poderá solicitar que determinadas funcionalidades sejam adicionadas, modificadas ou removidas.

Os requisitos do *software* também poderão sofrer mudanças durante a etapa de desenvolvimento, onde problemas de prazos, custos ou implementação, por exemplo, podem forçar a equipe de desenvolvimento a alterar os requisitos a fim de contornar o problema encontrado.

3.2 Tecnologias Adotadas

Muitos *softwares* que disponíveis na atualidade foram desenvolvidos utilizando os conceitos mais básicos de programação, geralmente um editor básico de textos e um compilador de uma determinada linguagem de programação. Inúmeras plataformas, *frameworks* e bibliotecas foram desenvolvidos para possibilitar que o programador pudesse desenvolver *softwares* de forma mais eficiente a partir do conceito de reuso proposto na Engenharia de *Software*. Hoje existem tecnologias famosas como *Unity 3D*[4] e *Unreal Engine*[5] para a criação de jogos, Java[6] e .NET[7] para desenvolvimento de aplicações, por exemplo.

3.2.1 Plataforma de desenvolvimento .NET *Framework*

Concebido como uma plataforma de desenvolvimento de propósito geral, o .NET *Framework* da Microsoft Corporation permite que todo o código gerado seja executado em qualquer sistema que possua o componente de Tempo de Execução instalado (MICROSOFT CORPORATION, 2016).

---

[4] https://unity3d.com

[5] https://www.unrealengine.com

[6] https://www.java.com

[7] http://dot.net

O .NET permite que *softwares* sejam escritos em várias linguagens diferentes e comuniquem entre si de forma nativa. Atualmente as linguagens *C#, F# e Visual Basic* são mantidas nativamente como parte do *framework*, mas além destas linguagens é possível adicionar outras através integrações de terceiros.

O .NET também apresenta recursos avançados como *GC* (*Garbage Collector* - Coletor de Lixo) para gerência automática de memória, tipos seguros, expressões lambda, delegados, tipos genéricos, programação assíncrona, LINQ (*Language Integrated Query* – Linguagem integrada de consulta), interoperabilidade nativa, dentre outros (MICROSOFT CORPORATION, 2016).

Quando um programa escrito em .NET é compilado, o código não é convertido diretamente para código executável, mas para uma outra linguagem chamada MSIL (*Microsoft Intermediate Language* – Linguagem Intermediária da Microsoft), ou simplesmente IL. A MSIL é uma linguagem composta de instruções independentes de CPU, o que permite convertê-la eficientemente em código nativo. Antes que o *software* possa ser executado é necessário que o código MSIL seja convertido para código nativo, tarefa feita pelo Compilador *Just-In-Time*. O compilador *Just-In-Time* é um componente do .NET responsável por realizar a tradução da MSIL para código nativo específico da arquitetura da CPU em tempo de execução do *software* (MICROSOFT CORPORATION). A Figura 1 apresenta o diagrama da arquitetura e componentes do .Net.

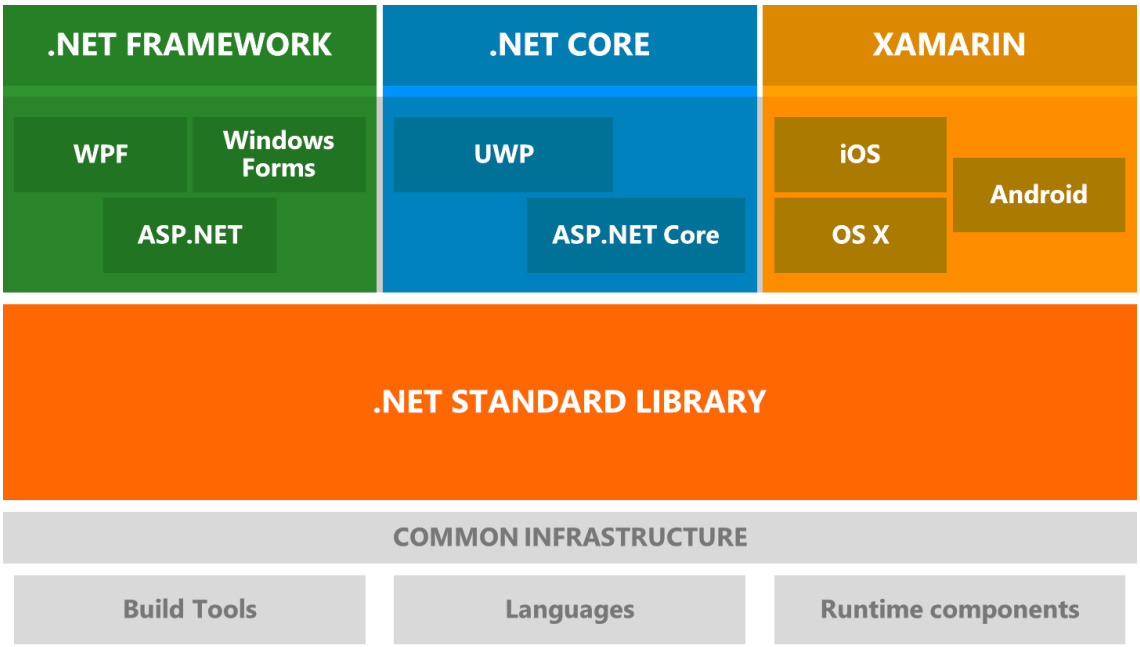


Figura 1: Componentes do .NET[8]

[8] Fonte: https://docs.microsoft.com/pt-br/dotnet/articles/standard/components

3.2.2 Linguagem de Programação C# (C *Sharp*)

Lançada em julho de 2000, o C# (pronunciado “C-Sharp”) é uma linguagem de programação orientada a objetos desenvolvida pela Microsoft Corporation. O C# possui uma sintaxe muito similar ao C++ e Java, o que possibilita uma compreensão e aprendizado mais fácil da linguagem por programadores experientes em C/C++, Java e similares. Seus principais inventores são Anders Hejlsberg, Scott Wiltamuth, e Peter Golde (ECMA INTERNATIONAL, 2006).

Atualmente a linguagem encontra-se na versão 6 (MARK MICHAELIS, 2014), e conta com recursos avançados de programação, como LINQ (*Language-Integrated Query*), que é uma implementação do paradigma de programação declarativo para a linguagem (MICROSOFT CORPORATION, 2015), métodos assíncronos, expressões Lambda, entre outros (ECMA INTERNATIONAL, 2006).

A principal motivação para a adoção da linguagem de programação C# como linguagem principal de desenvolvimento para o projeto se deve a conhecimentos adquiridos anteriormente com projetos em *ASP.NET* e *Silverlight* para a web, *Windows Mobile* 6 e *Windows Phone 7/8/10* para dispositivos móveis, *Windows Forms*, *Console* e *Windows Presentation Foundation (WPF)* para Desktop, e, *XNA Framework* para desenvolvimento de jogos. Todas essas tecnologias, além de serem desenvolvidas pela Microsoft, possuem também como denominador comum, a possibilidade de se programar utilizando C#.

3.2.3 *Windows Presentation Foundation* (WPF)

O *Windows Presentation Foundation* ou WPF é um *framework* para a criação de aplicações clientes que possibilitem experiências visualmente incríveis para os usuários. O WPF conta com um conjunto abrangente de recursos de desenvolvimento de aplicativos que incluem XAML (*Extensible Application Markup Language*), controles, associação de dados, gráficos 2D e 3D, animações, documentos, mídia, texto, etc. O WPF é um recursos presente no .NET Framework para que aplicações possam fazer uso das classes do .NET *Framework* (Introduction to WPF). Em outras palavras, o WPF é um conjunto de recursos de desenvolvimento que permitem desenvolver *softwares* com interfaces gráficas ricas para o usuário.

Embora uma interface gráfica possa ser totalmente construída no código da aplicação, o WPF disponibiliza uma linguagem de marcação denominada XAML, que simplifica a criação de telas, controles, comportamentos, animações, entre outros. Seu grande

diferencial é a capacidade de extensibilidade através das classes do *software*. Para utilizar uma classe escrita em C#, por exemplo, basta referenciar o pacote da classe (*namespace*) atribuindo um prefixo único no XAML. Em seguida basta criar uma nova etiqueta (*tag*) com o prefixo do *namespace* seguido do nome do controle desejado. A Figura 2 apresenta um exemplo de elemento estendido utilizando XAML.

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:c="clr-namespace:Archidata.Core.Controls;assembly=Archidata.Core" >

    <c:FlyoutMenu/>

</Window>
```

Figura 2 Exemplo de elemento estendido via XAML (FlyoutMenu)

3.3 Padrões de Projeto

Segundo Sommerville (SOMMERVILLE, 2007), um padrão de projeto é uma descrição do problema e a essência da solução, dessa forma a solução pode ser reusada em aplicações diferentes.

Padrões de projeto são geralmente empregados para se resolver um problema encontrado durante a etapa de desenvolvimento de *software*. Porém, alguns padrões de projeto exigem que seu uso parta desde o início do desenvolvimento do *software*, já que determinados padrões podem ditar toda a estrutura ~~com~~ que o *software* deverá possuir, como por exemplo, o padrão MVVM.

Em suma, padrões de projeto são maneiras de descrever melhores práticas, bons projetos, e captam a experiência de uma maneira possível de ser reusada por outros (SOMMERVILLE, 2007).

3.3.1 MVVM

O padrão de projeto MVVM (*Model-View-ViewModel*) é o principal padrão de projeto utilizado no desenvolvimento do *software* proposto neste trabalho. O padrão de projeto MVVM propõe uma organização para todo o projeto, sendo dividido em três partes principais: *Model*, *View* e *ViewModel*. O padrão MVVM é comumente utilizado em *softwares* escritos para a plataforma .NET que fazem uso de XAML para layout (MICROSOFT CORPORATION, 2016).

O padrão de projeto MVVM é composto basicamente por três elementos, sendo eles, ***Model***: O modelo utilizado no MVVM é uma implementação do modelo de domínio do *software* que inclui um modelo de dados, juntamente com as regras de negócio e validação dos dados. ***View***: A visão é responsável por definir a estrutura, layout e aparência do que o usuário vê na tela. A visão em MVVM é sempre definida utilizando XAML. ***View-Model***: O Modelo de visão atua como um intermediário entre a visão e o modelo, e é responsável por lidar com a lógica de exibição (MICROSOFT CORPORATION, 2016).

A Figura 3 mostra a estrutura e fluxo de dados do padrão MVVM.

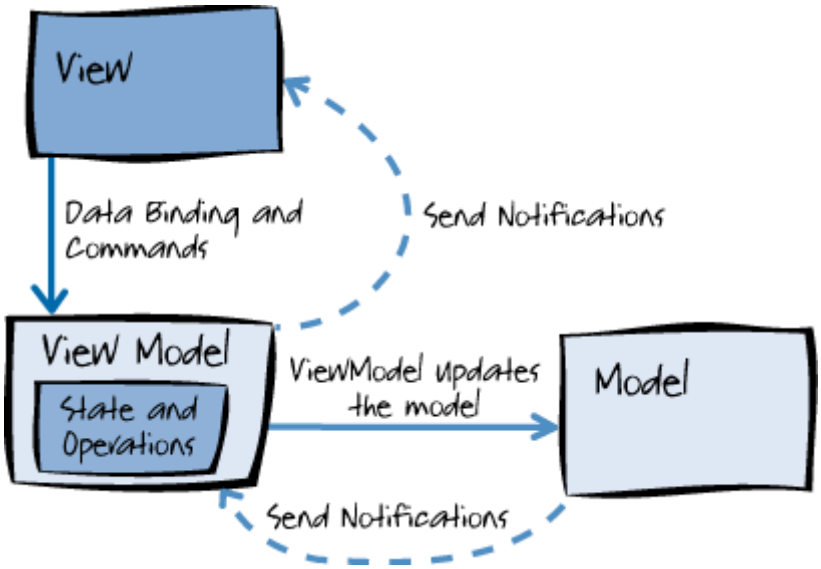


Figura 3: Estrutura do padrão MVVM[9].

3.3.2 *Memento*

Memento é utilizado na implementação do *plug-in* do diagrama Entidade-Relacionamento para permitir que as funcionalidades de desfazer e refazer pudessem ser adicionadas ao *plug-in*. Basicamente, a cada alteração que o usuário faz no diagrama em que está trabalhando, uma cópia do exato estado do diagrama é feita e logo em seguida colocada em uma estrutura de pilha.

Sempre que o usuário necessitar desfazer uma ação, o estado do topo da pilha responsável pelo desfazer é restaurado e o estado atual é colocado na pilha responsável pelo

[9] Fonte: https://msdn.microsoft.com/en-us/library/hh848246.aspx?f=255&MSPPError=-2147217396

refazer. Sempre que um novo estado é colocado na pilha responsável pelo desfazer, a pilha responsável pelo refazer deve ser esvaziada. Um estado só pode ser colocado na pilha responsável pelo refazer quando uma ação de desfazer for acionada.

3.3.3 *Observer*

Uma característica presente na plataforma .NET é a interface *INotifyPropertyChanged*. Esta interface é responsável por notificar a todos os componentes que fazem referência a um determinado objeto que o valor de uma ou mais de suas propriedades foram alteradas. Isso permite que os componentes gráficos modifiquem de forma automática o valor referenciado sem a necessidade de se criar rotinas para isso. Além disso, a coleção de dados *ObservableCollection*<T> garante que toda modificação feita diretamente nesta estrutura seja refletida imediatamente em todos os elementos de interface gráfica que fazem uso desta estrutura. A Figura 4 demonstra como a interface *INotifyPropertyChanged* foi utilizada para notificar sempre que a propriedade *Name* tiver seu valor modificado.

```
public class Attribute:INotifyPropertyChanged
{
    private string name;
    public string Name
    {
        get { return name; }
        set { name = value; OnPropertyChanged("Name"); }
    }
    public event PropertyChangedEventHandler PropertyChanged;

    private void OnPropertyChanged(string property)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(property));
    }
```

Figura 4: Exemplo de uso da interface *INotifyPropertyChanged*

3.4 Estrutura e arquitetura do *software*

Para que o *software* atenda aos requisitos funcionais, é necessário que ele possua uma organização estrutural interna que possibilite cumprir tais requisitos. Pressman (PRESSMAN, 2011) define projeto de arquitetura como a representação da estrutura dos dados e os componentes de programa necessários para se construir um sistema computacional. A Figura 5 exibe graficamente a estrutura proposta para o *software*.

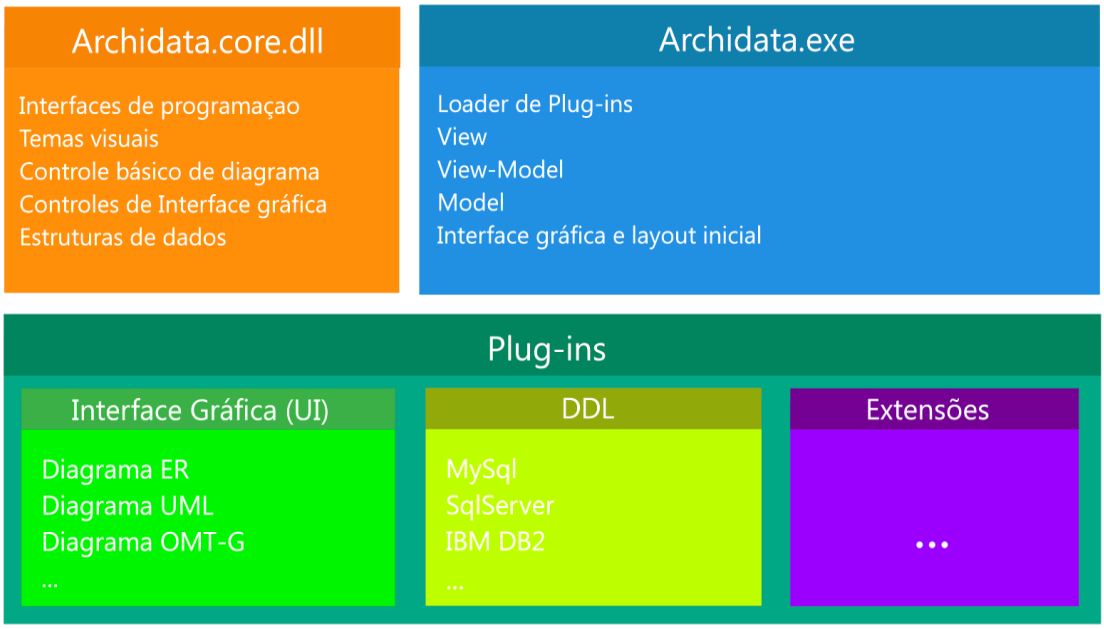


Figura 5: Estrutura do *software* proposto

A DLL (*Dynamic Link Library*) ou biblioteca de vínculo dinâmico Archidata.core.dll contém todas as classes e interfaces de programação comuns a todos os *plug-ins*. Através dessa DLL cada *plug-in* pode se comunicar com o executável da aplicação e vice-versa. Os *plug-ins* foram inicialmente divididos em três tipos:

- **Interface Gráfica:** *Plug-ins* exibidos graficamente para o usuário final, como por exemplo, diagramas, editores, terminais e diálogos.
- **DDL (*Data Definition Language*):** São *plug-ins* responsáveis pela comunicação com SGBDs e sua geração de código SQL. Armazenam a DDL do SGBD, ou seja, as características individuais de cada SGBD necessárias para a geração de código SQL (ISO/IEC 9075-11:2011, 2011).
- **Extensões:** *Plug-ins* genéricos capazes de atuar diretamente com os dados de outros *plug-ins*. Foram idealizados para suprir quaisquer outras necessidades não previstas inicialmente na aplicação.

É bastante comum pressupor o uso de DLLs do .NET, ao lidar com a ideia de carregar conteúdo dinamicamente, já que seu uso é bastante típico na plataforma. É comum utilizar DLLs no .NET para adicionar funcionalidades como controles de interface gráfica, ou utilizar bibliotecas para o processamento de determinado tipo de dado. Em contrapartida, *softwares* que fazem uso de DLLs dessa forma não fazem o carregamento dinâmico das bibliotecas, ou seja, quando o *software* foi compilado, a aplicação já possuía todos os dados necessários para acessar as funcionalidades da biblioteca.

No atual cenário da aplicação, o uso estático de bibliotecas se torna uma opção inviável, já que seu uso está condicionado à recompilação da aplicação principal sempre que for necessário adicionar um novo recurso. Para contornar este problema, foi utilizado a biblioteca MEF (*Managed Extensibility Framework).* O MEF é uma biblioteca nativa do .NET que não depende de nenhuma instalação para uso nos *softwares* escritos para a plataforma. O MEF permite que DLLs sejam carregadas em tempo de execução pela aplicação.

Embora o carregamento dinâmico oferecido pelo MEF possibilite contornar o problema anteriormente encontrado, ainda existe o problema de como a aplicação deverá proceder para comunicar com as DLLs de *plug-ins*, já que ela pode não conhecer a estrutura da DLL. O MEF propõe o uso de herança e/ou polimorfismo juntamente à criação de uma DLL específica para este objetivo.

Em suma, um conjunto de interfaces, classes ou classes abstratas podem ser definidas nessa DLL. A DLL criada deverá ser referenciada na aplicação, permitindo assim conhecer os comandos que obrigatoriamente estarão contidos em cada *plug-in* criado para a aplicação. Assim como a aplicação, cada novo *plug-in* deverá referenciar a DLL criada e implementar ou herdar a interface ou classe de acordo com o tipo respectivo do *plug-in*.

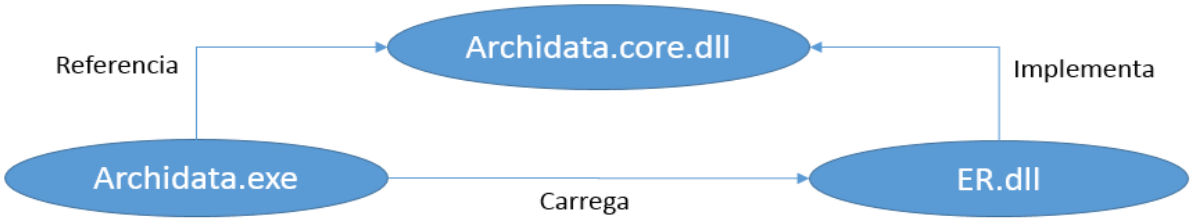


Figura 6: Estrutura MEF para o *plug-in* ER

De acordo com Figura 6, a DLL Archidata.core.dll é responsável por manter as interfaces de programação comuns a cada *plug-in*, onde tanto Archidata.exe e demais *plug-ins* deverão referenciar esta DLL. Dessa forma é possível que o *software* se comunique corretamente com cada um dos *plug-ins*, já que estes deverão implementar a interface de programação correspondente ao tipo do plug-in.

3.5 *Design* de Interface Gráfica

A interface gráfica é um importante componente responsável pela interação com o usuário. Segundo Pressman (PRESSMAN, 2011), o projeto de interfaces com o usuário cria um meio de comunicação efetivo entre o ser humano e o computador. Um bom projeto de interface pode ser o fator determinante para a aceitação do produto pelo usuário final.

3.5.1 *Splash Screen* (Tela de Carregamento)

Uma *Splash Screen* (tela de carregamento), é um importante componente na interação com o usuário. Através dela é possível saber que o *software* está realizando alguns processamentos antes de abrir a interface gráfica principal da aplicação. As telas de carregamento podem exibir o progresso e reportar ao usuário qual tarefa está realizando. É bastante comum utilizar animações para mostrar ao usuário que a aplicação está trabalhando.

Consiste de uma pequena janela que se abre ao executar a aplicação, cujo propósito é meramente informativo. As telas de carregamento podem exibir controles que possibilitem que o usuário interaja com o *software* durante o carregamento, mas esta característica não é obrigatória. Ao terminar o carregamento, a *Splash Screen* é finalizada e em seguida, a interface gráfica da aplicação é exibida para o usuário.

A Figura 7 demonstra o *design* da tela de carregamento projetada para o *software* proposto. A tela de carregamento foi desenvolvida utilizando o editor de imagens gratuito *The Gimp*[10].

Figura 7: Tela de Carregamento para o *software* proposto

3.5.2 Tela Principal

Para o *design* da tela principal da aplicação, utilizou-se o *Microsoft Office Ribbon.* A principal motivação para esta escolha foi o fato de que a interface gráfica proposta pela Microsoft já possuía uma grande aceitação pelos seus usuários desde seu lançamento para o Microsoft Office 2007.

Mesmo sendo propriedade intelectual da Microsoft, o uso da interface gráfica do *Office* foi inicialmente liberado para uso comercial ou não comercial, e, em qualquer sistema operacional, desde que estivesse associado a uma licença que podia ser obtida gratuitamente através do site da Microsoft (HARRIS, 2006). Atualmente não é mais necessário adquirir uma

[10] https://www.gimp.org/

licença para se utilizar a interface, e a página do site da Microsoft destinada a obtenção de licenças de uso foi removida (MICROSOFT CORPORATION). A Figura 8 apresenta o *design* final do *software* proposto, contendo um conjunto de abas superiores destinadas às opções dos *plug-ins* carregados em tempo de execução; uma barra lateral contendo um navegador de arquivos e editor de propriedades; e a área de trabalho do *software*.

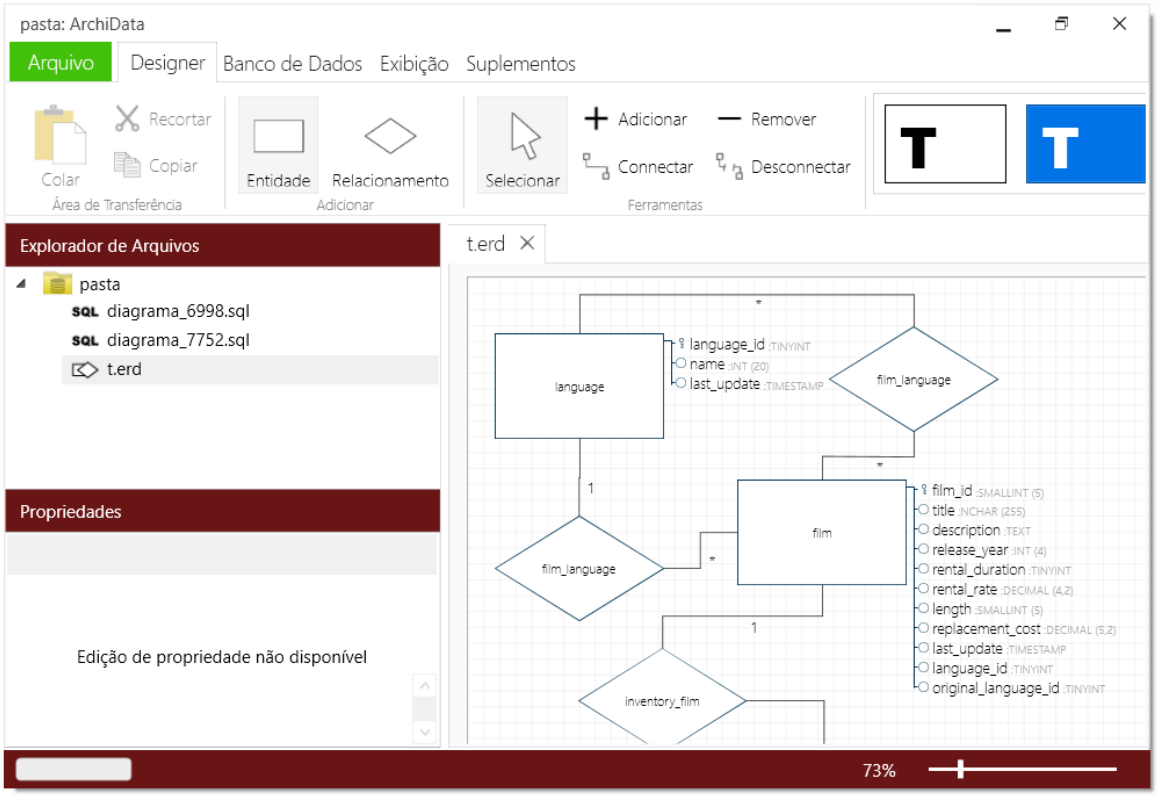


Figura 8: Interface gráfica final da tela principal

A interface gráfica proposta possui alguns controles pré-fixados com os quais só é possível interagir após a abertura de um arquivo em um *plug-in*. Quando um *plug-in* é carregado, algumas opções (como *Zoom*, Abrir Arquivo, Salvar, Imprimir, Copiar, Colar) se tornam ativas para o usuário. Essas opções podem ser desabilitadas pelo próprio *plug-in* caso não se deseje interagir com essas opções em determinado momento.

Para que um *plug-in* possa desabilitar a interação com uma determinada opção, foram criadas propriedades booleanas (verdadeiro ou falso) para cada uma destas opções. Quando o plug-in fez uso de herança para a classe base do tipo do plug-in, estas propriedades foram automaticamente adicionadas à classe do plug-in. Para desabilitar uma opção basta atribuir o valor correspondente à opção para *false*, por exemplo, a opção imprimir possui a propriedade *CanPrint*, logo, caso o plug-in seja capaz de imprimir, define-se o valor da propriedade para *true*, do contrário *false*.

3.5.3 *Plug-in* Diagrama Entidade-Relacionamento

O plug-in Diagrama Entidade-Relacionamento é um plug-in destinado a permitir que o usuário realize diretamente uma modelagem conceitual de banco de dados utilizando o modelo Entidade-Relacionamento (ER) proposto por Peter Chen (CHEN, 1976).

Para que seja possível modelar um diagrama no *plug-in*, é necessário que este disponibilize um conjunto de ferramentas capazes de modificar o estado do diagrama. Como apresentado na Figura 9, o *plug-in* disponibiliza algumas ferramentas básicas (como copiar, recortar e colar) que são comandos comuns a todos os *plug-ins* do *software*, e, ferramentas mais específicas à modelagem (como selecionar, adicionar, conectar).

Figura 9: Conjunto de ferramentas expostas pelo *plug-in*

O *design* do diagrama foi baseado no *design* da ferramenta **BrModelo** (CÂNDIDO, 2007), com algumas modificações para reduzir o tempo de implementação do *plug-in*. Por motivos didáticos, os atributos de chave primária receberam um ícone de chave, conforme descrito na Figura 10.

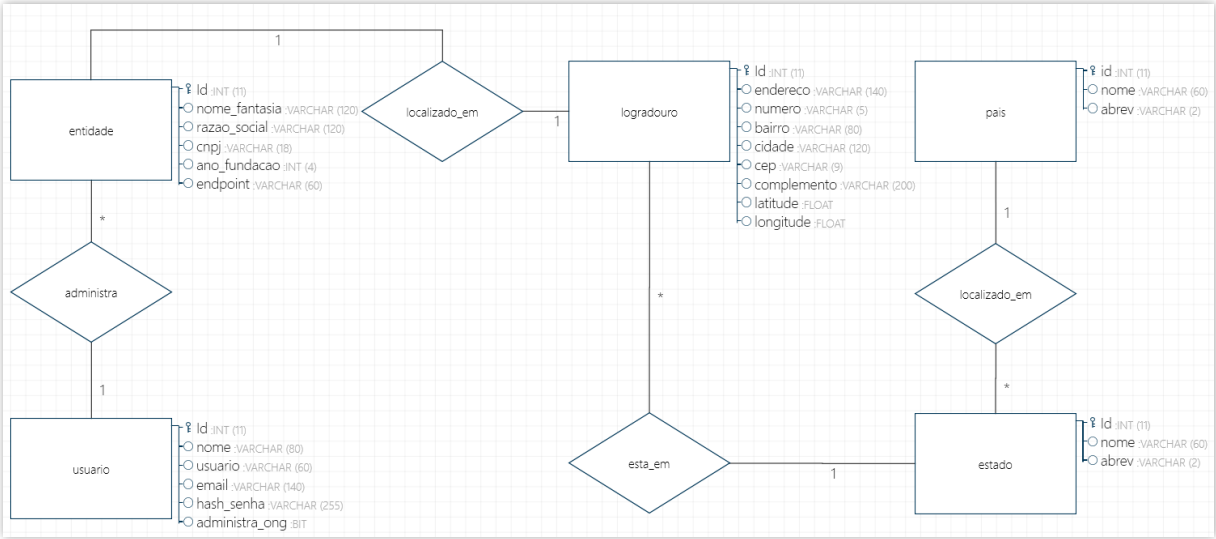


Figura 10: Exemplo de Diagrama criado no *plug-in* ER

Cada um dos atributos é exibido em uma lista do lado direito de cada entidade, garantindo assim uma melhor organização de atributos. Além disso, os atributos são sempre seguidos do tipo de dado e tamanho escolhidos para ele. Esta abordagem tem como principal objetivo, facilitar a percepção do usuário de possíveis erros cometidos que possam gerar uma saída não esperada pelo usuário. Isso se torna mais evidente durante a geração de código SQL, onde determinado campo poderá acabar por receber um tipo ou tamanho incorreto, cujo erro se tornará mais evidente durante a criação do banco de dados.

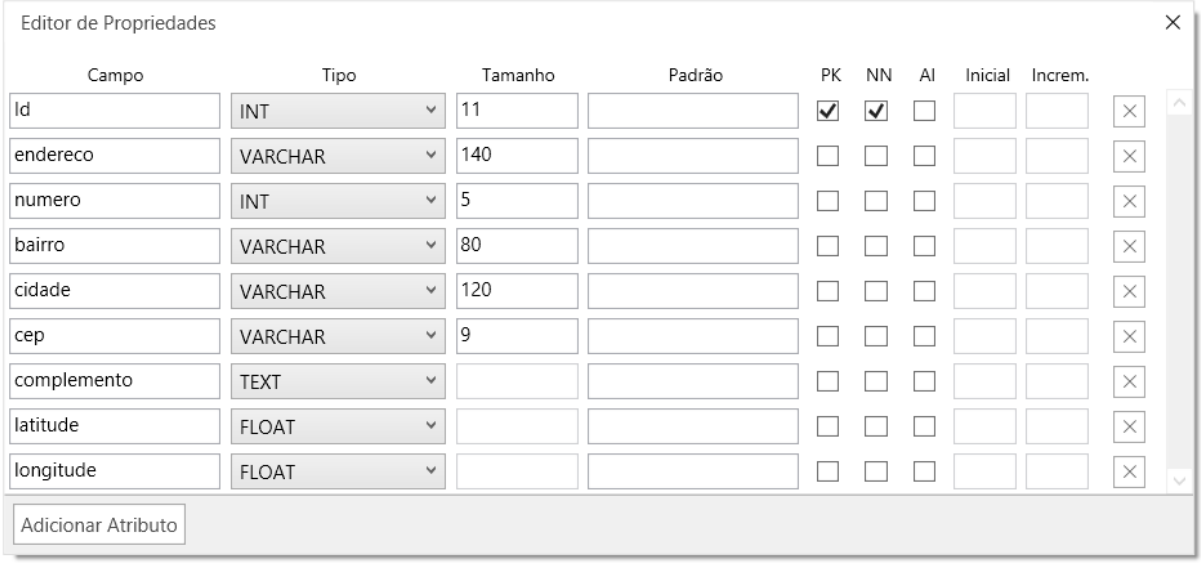


Figura 11: Editor de propriedades do *plug-in*

O *plug-in* também conta com um editor de atributos para as entidades. O editor foi projetado de forma a abranger o maior número de SGBDs baseado na interseção das características dos principais SGBDs. O editor consiste de uma janela flutuante que se abre sobre o diagrama, conforme visto na Figura 11. Apesar de haver uma certa liberdade para se programar um *plug-in*, um conjunto de temas visuais foram disponibilizados juntamente à DLL Archidata.core.dll para que todo o *software* siga o mesmo *design*, seu uso não é obrigatório, mas é fundamental para se manter uma experiência adequada ao usuário.

3.5.4 *Plug-in* editor de código SQL

O *plug-in* do editor de código SQL consiste de um editor básico de textos com alguns recursos adicionais que possibilitam ao usuário uma melhor experiência durante a edição de código. O *plug-in* conta com destaque sintático, ou seja, algumas palavras presentes na linguagem SQL são automaticamente grafadas em uma determinada cor à medida que o usuário escreve código, conforme descrito na Figura 12. O editor também conta

com o recurso de auto sugerir, permitindo que o usuário obtenha sugestões de complementação de código sempre que ele pressionar CTRL + Espaço.

diagrama_5888.sql

```sql
CREATE DATABASE teste;
USE teste;

CREATE TABLE `logradouro`(
    `Id` INT(11) NOT NULL,
    `endereco` VARCHAR(140),
    `numero` VARCHAR(5),
    `bairro` VARCHAR(80),
    `cidade` VARCHAR(120),
    `cep` VARCHAR(9),
    `complemento` VARCHAR(200),
    `latitude` FLOAT,
    `longitude` FLOAT,
    PRIMARY KEY (`Id`)
);

CREATE TABLE `estado`(
    `Id` INT(11) NOT NULL,
    `nome` VARCHAR(60),
```

Figura 12: *Plug-in* editor de código SQL

O desenvolvimento geral da aplicação procurou não utilizar muitos componentes de terceiros, porém, a biblioteca de código aberto *AvalonEdit* possuía um conjunto muito grande de funcionalidades e um elevado nível de maturação. O *AvalonEdit* foi criado para ser o editor de código-fonte da IDE (Ambiente de Desenvolvimento Integrado) *Sharp Develop*, que é uma poderosa alternativa ao Visual Studio da Microsoft (GRUNWALD).

Como o componente *AvalonEdit* se mostrou um poderoso e eficiente editor de código-fonte, o *plug-in* de edição SQL foi composto a partir da integração entre este componente e o *plug-in*.

O componente AvalonEdit também possibilita a inserção de marcações sublinhadas para informar ao usuário erros sintáticos encontrados no código digitado. Embora esta característica não esteja presente na atual versão do *plug-in*, em futuras versões um analisador léxico e sintático poderão ser implementados e facilmente integrados ao componente.

4 ARCHIDATA

Figura 13: Ícone da aplicação

O produto final foi batizado de **ARCHIDATA** (Figura 13), que consiste da aglutinação das palavras Arquiteto e Dados em inglês (*Architect* + *Data*), como forma de expressão da sua principal funcionalidade: arquitetar dados.

O código fonte assim como o *software* são atribuídos à Licença Pública Geral versão 3 (*General Public Licence v3*), ou **GPLv3**, como forma de garantir contínua colaboração e continuidade do projeto, além de garantir que o *software* sempre será livre. A GPLv3 exige que o *software* ou qualquer obra derivada seja distribuída juntamente ao seu código-fonte.

Os principais benefícios encontrados na adesão do *software* desenvolvido incluem:

- **Gratuito e de código aberto**: O uso do *software* é livre em projetos individuais ou comerciais. Seu código fonte também está disponível para consultas, modificações e adaptações para projetos públicos, comerciais ou privados desde que seu código fonte seja distribuído juntamente ao produto.
- **Suporte a múltiplos SGBDs**: A ferramenta foi arquitetada de modo a permitir que todo o trabalho de projeto nela realizado pudesse ser exportado para todos os SGBDs. Para isso a ferramenta inclui suporte a extensões DDL.
- **Suporte a múltiplos diagramas e notações**: Embora a atual versão do *software* só possua suporte a diagramas Entidade-Relacionamento (ER), novos plug-ins baseados em outras notações podem ser criados e disponibilizados.
- **Engenharia reversa**: Modelos podem ser obtidos a partir da estrutura de bancos de dados já existentes, permitindo assim modifica-los e obter novamente o código de criação atualizado.
- **Editor de código SQL**: Permite escrever e editar códigos SQL diretamente no *software*, com auxílio de destaque sintática e auto sugerir.

O *software* desenvolvido também foi submetido a uma sequência de testes como forma de validação e verificação, conforme descrito nos capítulos seguintes.

5 Teste de *Software*

Como uma importante etapa do processo de desenvolvimento de *software*, os testes são uma etapa fundamental para a validação e verificação de um *software*. Um teste bem projetado deve encontrar a maioria dos erros em um *software*, garantindo assim, uma melhor experiência para o usuário final ao utilizá-lo (PRESSMAN, 2011).

Existe uma série de tipos de testes projetados para avaliar diferentes aspectos de um *software*. Embora pareça ideal aplicar todos os testes possíveis a um *software*, na maioria dos casos, esta possibilidade é inviável. Os testes devem ser projetados de forma a revelarem o maior número de erros de um *software*, logo todo essa etapa de projeto de testes demanda tempo e dinheiro. Na maioria dos casos é inviável o uso de testes exaustivos, já que o número de caminhos lógicos em um *software* pode demandar um tempo computacional muito alto, inviabilizando totalmente seu uso (PRESSMAN, 2011).

5.1 Teste de unidade

Os testes de unidade, como o próprio nome sugere, são designados a testar cada unidade de um projeto, no caso de linguagens orientadas a objetos, cada classe. Segundo Pressman (PRESSMAN, 2011), o teste de unidade focaliza o esforço de verificação na menor unidade de projeto do *software*. Em um projeto .NET, os testes são programados em projetos separados que importam cada uma das classes do projeto original já compiladas, e, invocam separadamente cada um de seus métodos com base no código programado para o teste.

Cada método de uma unidade testada pode requerer que sejam programados testes diferentes dos demais métodos da classe. Além disso, algumas classes requerem que alguns de seus atributos sejam inicializados antes da execução dos testes. Para garantir que essa inicialização seja feita, o *framework* de testes de unidade do .NET dispõe um atributo chamado *TestInitialize*. O atributo *TestInitialize* deve ser colocado acima de qualquer método de inicialização criado no teste, desta forma, quando o teste for executado, o método de inicialização será chamado antes dos métodos de teste.

Da mesma forma que certos testes exigem uma inicialização, alguns testes também poderão executar métodos após a execução dos métodos de teste. Um exemplo desse tipo de teste é a comunicação com banco de dados que exige que sua comunicação seja encerrada corretamente, ou simplesmente liberar memória principal ou secundária após a execução dos testes. Para que isso seja possível o *framework* de teste disponibiliza o atributo *TestCleanup*, que deverá ser colocado sobre os métodos que necessitem execução

após a finalização dos testes. A Figura 14 apresenta um exemplo de classe de teste de unidade.

```
[TestClass()]
public class EditorTests
{
    [TestInitialize]
    public void InitializeTests()
    {
        //código de inicialização aqui
    }

    [TestMethod()]
    public void UndoTest()
    {
        //código de teste aqui
    }

    [TestCleanup]
    public void Clean()
    {
        //código de limpeza e finalização aqui
    }
}
```

Figura 14: Exemplo de classe de teste de unidade

5.1.1 Executando testes de unidade no *software*

Ao projetar testes de unidade para o *software* é possível perceber que a divisão do projeto como um todo em projetos individuais torna a criação de testes muito mais simples. Como cada DLL do *software* é criada em um projeto a parte dentro do ambiente de desenvolvimento, cada teste de unidade também irá produzir um novo projeto respectivamente. O código de cada teste fica então separado dos testes dos demais projetos. Cada projeto de teste pode então, focar em testar cada um dos componentes do projeto alvo.

A maioria dos testes escrito para os componentes do *software* previa testar funcionalidades e chamadas e funções passando sempre como parâmetro um conjunto básico de entradas válidas e um conjunto de entradas inválidas. Quando os testes eram executados, o ambiente de desenvolvimento reportava quais métodos passaram no teste e quais foram reprovados através da notação da letra V na cor verde e a letra X na cor vermelho respectivamente.

A Figura 15 apresenta o resultado para os testes do componente de edição de código SQL do *software*. Os resultados apontados também apresentam o tempo gasto na execução de cada um dos métodos de teste, onde em alguns casos, um tempo limite (*timeout*) é necessário para que um teste seja reprovado caso sua execução exceda o tempo aceitável determinado nos documentos do projeto.

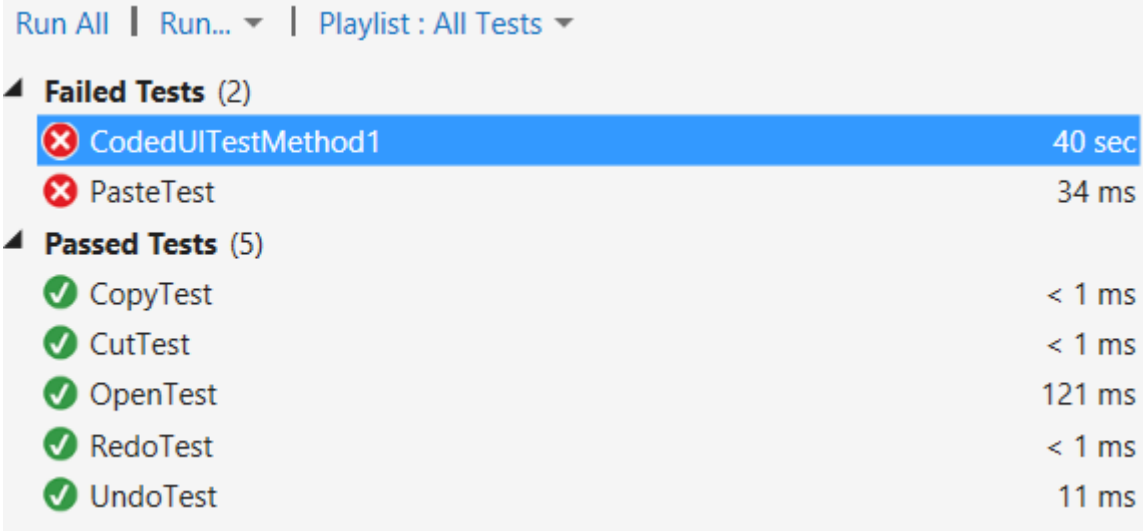


Figura 15: Exemplo de resultado de teste reportado pelo ambiente de desenvolvimento

A etapa de testes de unidade se deu como encerrada quando todos os erros encontrados nos testes foram corrigidos. A partir deste ponto, alguns novos erros foram descobertos em testes posteriores, que também foram corrigidos. Quando os testes não encontraram mais erros, os testes de unidade foram encerrados para a versão corrente do *software*, ficando a cargo de outros tipos de teste descobrir novos erros.

5.2 Teste *Alfa*

Os testes *alfa* são um tipo de teste de caixa preta destinados a testar diretamente a interação do usuário com o produto. Este teste permite descobrir fatores que podem passar despercebidos para o programador, mas que são vitais para uma experiência de usuário adequada.

Normalmente os testes alfa são conduzidos em um ambiente controlado, onde algumas ferramentas podem ser utilizadas para capturar desde aspectos básicos da interação do usuário com o *software*, assim como aspectos mais sutis. Câmeras podem ser utilizadas para capturar expressões faciais do usuário que indiquem dificuldade de utilização. Da mesma forma, frases ditas pelo usuário podem ser capturadas por um microfone para serem avaliadas pelo teste (PRESSMAN, 2011).

Mas talvez o ponto mais interessante seja a captura da imagem da tela do próprio computador. Essa captura permite analisar o comportamento esperado do usuário juntamente ao comportamento obtido na execução dos testes. A captura da tela, sincronizada com os demais recursos, como câmeras e microfones, permitem uma análise completa da interação do usuário.

Mesmo que várias ferramentas facilitem a execução de testes alfa, a presença de um avaliador poderá permitir a percepção de pequenos fatores que podem passar despercebidos por estas ferramentas. Desta forma, o avaliador deverá ser neutro durante a condução dos testes, não fornecendo nenhum tipo de ajuda ao usuário testador.

5.2.1 Ambiente de Testes

Para a condução do teste da aplicação, foi reservado um laboratório de informática em que o *software* foi instalado previamente. Como o público alvo da aplicação requer que seus usuários possuam um conhecimento mínimo em modelagem de bancos de dados, um grupo de testes de 19 alunos da disciplina de Banco de Dados I foi convidado a participar dos testes, sendo que 18 compareceram efetivamente no dia do teste. Um formulário (APÊNDICE B) contendo uma descrição um pouco mais técnica do teste foi distribuída aos usuários, onde caso este concordasse com os termos descritos, assinava abaixo confirmando a participação no teste.

Para a captura das interações dos usuários, foi utilizado a ferramenta gravador de passos do Windows (psr.exe). O gravador de passos do Windows é uma ferramenta nativa do sistema operacional Microsoft Windows. Como o próprio nome sugere, o gravador de passos do Windows está presente no sistema operacional como uma ferramenta para reprodução passo a passo de problemas.

O gravador de passos do Windows realiza a captura da tela do usuário a cada interação feita, como por exemplo, clicar com o mouse, digitar um texto, etc. O gravador de passos do Windows produz um arquivo html contendo uma série de imagens de cada interação do usuário. Mesmo sendo o gravador de passos do Windows, uma ferramenta de reprodução de problemas, ainda se mostra uma ferramenta útil para a captura da interação básica do usuário.

5.2.2 Condução dos Testes

O experimento realizado no laboratório consistiu de três problemas (APÊNDICE D), que tinham como objetivo, explorar o maior número de funcionalidades do *software* pelo usuário.

Antes do início dos testes, foi explicado para cada usuário que a intenção do experimento não era avaliar se os diagramas produzidos na ferramenta estavam corretos, mas a forma com que a ferramenta era utilizada. Além disso, por se tratarem de alunos da disciplina de Banco de Dados, foi necessário informar que a participação nos testes não iria

de forma alguma impactar na nota do aluno na disciplina já que a participação era livre, ou seja, caso algum aluno não quisesse participar, tinha total liberdade para se retirar.

Em alguns dos problemas propostos para resolução, o usuário era responsável pela própria descoberta das funcionalidades necessárias para a solução do problema disponibilizado. O principal objetivo desta abordagem era avaliar o grau de dificuldade com que um usuário poderia vir a ter durante a utilização do *software* para encontrar um recurso necessário.

O experimento contou com a participação de dois avaliadores (o professor da disciplina de banco de dados e o autor deste trabalho). Cada um dos avaliadores contava com uma prancheta ~~de~~ contendo algumas questões – vide APÊNDICE C – que poderiam vir a ser observadas durante o experimento. Embora as questões abordassem uma ampla gama de possibilidades, o avaliador também tinha total liberdade para observar quaisquer outros pontos que julgasse importantes e anotá-los livremente na prancheta.

Ao final de cada experimento, o usuário recebia um formulário – vide APÊNDICE E – que continha um conjunto de questões cuja maioria tinha como objetivo quantificar o grau de satisfação, aceitação e entrosamento do usuário com a ferramenta. Todos os formulários continham ao final um campo de livre escrita para que o usuário pudesse registrar opiniões que não foram abordadas nas questões anteriores.

5.2.3 Resultados e Discussões do Teste Alfa

O experimento conduzido em laboratório permitiu uma coleta satisfatória de resultados que possibilitaram uma série de melhorias do *software*. Durante a execução dos testes, algumas exceções de *software* foram lançadas na interação do usuário com a aplicação. A Tabela 1 lista todas as exceções de *software* lançadas durante o teste.

Exceções Lançadas durante o Teste			
Tipo	Descrição	Causa no teste	Status
NullReferenceException	Ocorre na tentativa de se acessar um objeto cuja referência seja nula	Conectar duas entidades ou dois relacionamentos diretamente	Corrigido
		Conectar dois relacionamentos diretamente	Corrigido
		Conectar uma entidade ou relacionamento a si próprio	Corrigido
ArgumentNullException	Ocorre quando um método recebe um valor nulo não esperado como parâmetro	Deixar campos de atributos de entidade vazios	Corrigido

Tabela 1: Exceções lançadas durante o teste de *software*

Apesar do teste de unidade tentar abranger cada um dos aspectos individuais de cada classe, algumas características podem acabar não sendo tratadas durante os testes de unidade. Estas características não tratadas podem revelar um teste de unidade cuja eficiência não possua um desempenho desejado na descoberta de novos *bugs*. Em alguns casos, estes aspectos não tratados dependem de características não diretamente simuláveis no teste, como por exemplo, a interação do usuário, mais especificamente, a ordem de cada interação, que por sua vez, não depende da qualidade do teste de unidade programado.

Além das exceções capturadas durante a execução dos testes, os formulários de avaliação individual revelaram alguns aspectos e funcionalidades em que o *software* não possui implementação, ou a forma com que foi implementado não é a mais adequada para o usuário. Através dos resultados coletados nos formulários de avaliação individual (APÊNDICES F, G e H), a Tabela 2 foi formulada com as características que os usuários tiveram algum problema durante os testes. A tabela apresenta uma breve descrição do problema, a solução para o problema em nível de implementação de *software*, a prioridade de implementação da característica, e o status atual da implementação do *software*.

Problemas reportados pelos usuários			
Descrição	Solução	Prioridade	*Status*
O *Software* não questiona se deseja salvar quando fechado, fazendo todo trabalho se perder se não salvo anteriormente.	Implementar diálogo “deseja salvar?”	Alta	Corrigido
Encontrar o caminho para editar os atributos de uma entidade não é muito simples.	Adicionar clique duplo como alternativa.	Alta	Corrigido
Não é possível desfazer ou refazer uma ação.	Implementar CTRL + Z e CTRL + Y.	Alta	Corrigido
A janela de atributos não pode ser movida, impossibilitando de visualizar minhas alterações enquanto aberta.	Adicionar recurso de mover diálogos.	Média	Corrigido
O controle responsável por aplicar temas produz um comportamento inesperado ao passar o mouse sobre o controle.	Remover eventos de pré-visualização de temas para posterior correção.	Baixa	Solução Provisória
Definir o botão Enter como padrão para submissão de um formulário ou janela	Definir botão Ok como "Default" no XAML	Baixa	Corrigido
Incluir dicas ao passar o mouse sobre itens do *software*	Adicionar dicas (Tooltips) no XAML	Baixa	Parcialmente Corrigido
A exportação de um diagrama para imagem abrange toda a área de desenho ao invés de abranger somente a área desenhada	Adicionar janela com opções de customização de exportação	Baixa	Não Corrigido
Encontrar o caminho para renomear um objeto não é muito simples.	Renomear no próprio objeto	Baixa	Corrigido
Adicionar atributos a um relacionamento	Adicionar Modelo ER Estendido	Baixa	Não Adicionado

Tabela 2: Problemas relatados pelos usuários

6 Trabalhos futuros

Durante o desenvolvimento foi possível perceber que havia um conjunto de características inicialmente não prevista nos requisitos do software, já que sua presença não impossibilita a utilização do software. Embora não obrigatórias, estas características podem fornecer ao software, recursos que podem tornar a aplicação mais rica e consequentemente, trazer uma melhor experiência para o usuário.

A Tabela 3 lista as principais características que foram observadas durante o desenvolvimento do software.

Recurso	Justificativa
Plug-in diagrama UML	*Plug-in* com notação alternativa ao modelo conceitual Entidade-Relacionamento
Plug-in OMT-G e/ou UML GeoFrame	*Plug-in* para bancos de dados geográficos.
Analisador léxico e sintático para o editor SQL	Um analisador léxico e sintático permitiria analisar no próprio software o código SQL escrito pelo usuário.
Consultas SQL diretamente no editor SQL	Executar consultas diretamente a partir do código SQL escrito pelo usuário juntamente com a apresentação dos resultados.
Estender *plug-in* ER	Adicionar recursos presentes no diagrama Entidade-Relacionamento Estendido
Plug-ins de SGBDs	Adicionar suporte a novos SGBDs como PostgreSQL, Oracle, IBM DB2, SQL Server, SqLite, etc.

Tabela 3: Trabalhos futuros observados para o *software*

Alguns dos recursos previstos para trabalhos futuros tem como objetivo tornar a aplicação mais completa e independente de outros *softwares*. Ao executar consultas SQL dentro do próprio *software*, o usuário não terá necessidade de utilizar outros *softwares* para esta finalidade, por exemplo.

7 Conclusões

O uso de ferramentas na construção de novos sistemas computacionais é atualmente uma prática amplamente utilizada. Atualmente contamos com excelentes ferramentas que nos possibilitam construir *softwares* robustos sem grandes complicações de desenvolvimento.

Embora o *software* proposto ainda esteja em sua fase inicial, sua proposta de solução simples, gratuita e de código aberto o torna uma opção atrativa para desenvolvedores e organizações que não possuam grandes orçamentos para investimento em ferramentas comerciais. O código aberto permitirá aos usuários visualizar, discutir e colaborar em toda a extensão de código do *software*, agregando novas funcionalidades ou aperfeiçoando as funcionalidades já existentes.

Conforme o *software* sofre um aumento em suas linhas de código, fica cada vez mais evidente a necessidade da realização de um conjunto de testes cada vez mais robusto e preciso, capaz de encontrar o maior número de falhas possíveis. Como observou-se no teste *alfa*, o teste diretamente com o usuário final é também um importante fator que deve ser devidamente organizado de forma a não só encontrar erros na programação do *software*, mas também aspectos importantes para uma usabilidade ideal ao usuário.

O *feedback* obtido com os usuários durante os testes alfa permitiu descobrir uma série de fatores que pareciam inicialmente irrelevantes do ponto de vista do programador. Esses fatores se mostraram importantes a partir do momento em que o usuário interage diretamente com o *software,* revelando aspectos importantes para uma adequada experiência do usuário com o *software*, como por exemplo, o simples ato de mover um diálogo, que inicialmente não era possível.

O *software* aqui proposto conseguiu cumprir com todos os objetivos e requisitos inicialmente especificados para o projeto. Embora muitas das suas funcionalidades ainda necessitem de uma série de correções e melhorias, a ferramenta se mostrou bastante eficiente, satisfatória e visivelmente agradável na opinião do usuário final, o que se mostra um grande ponto positivo da ferramenta.

Referências

CÂNDIDO, C. H. brModelo: Ferramenta de Modelagem Conceitual de Banco de dados, 2007. Disponivel em: <http://www.sis4.com/brmodelo/monografia/monografia.htm>. Acesso em: 12 fev. 2017.

CHEN, P. P. **The Entity-Relationship Model--Toward a Unified View of Data**. [S.l.]: Massachussetts Institute of Technology, 1976.

CHU, S. Y. **Banco de dados:** organização, sistemas e administração. São Paulo: Atlas, 1990.

DATABASE Modelling Tools. **Database Answers**, 2015. Disponivel em: <http://www.databaseanswers.org/modelling_tools.htm>. Acesso em: 26 fev. 2017.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. **Introdução ao Teste de Software**. Rio de Janeiro: Elsevier, 2007.

ECMA INTERNATIONAL. C# Language Specification. **ECMA International**, 2006. Disponivel em: <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf>. Acesso em: 27 nov. 2016.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de banco de dados**. 6ª. ed. [S.l.]: Pearson Addison Wesley, 2011.

GRUNWALD, D. AvalonEdit. Disponivel em: <http://avalonedit.net/>. Acesso em: 18 fev. 2017.

HARRIS, J. Licensing the 2007 Microsoft Office User Interface. **Jensen Harris:** An Office User Interface Blog, 21 novembro 2006. Disponivel em: <https://blogs.msdn.microsoft.com/jensenh/2006/11/21/licensing-the-2007-microsoft-office-user-interface/>. Acesso em: 24 fev. 2017.

INTRODUCTION to WPF. **Microsoft MSDN**. Disponivel em: <https://msdn.microsoft.com/en-us/library/aa970268(v=vs.100).aspx>. Acesso em: 27 nov. 2016.

ISO/IEC 9075-11:2011. **International Organization for Standardization (ISO)**, dezembro 2011. Disponivel em: <https://www.iso.org/standard/53685.html>. Acesso em: 14 janeiro 2017.

MARK MICHAELIS. Como o C# 6.0 simplifica, esclarece e condensa o seu código. **Microsoft MSDN Magazine**, 2014. Disponivel em: <https://msdn.microsoft.com/pt-br/magazine/Dn879355.aspx>. Acesso em: 27 nov. 2016.

MICROSOFT CORPORATION. Introduction to LINQ Queries (C#). **Microsoft MSDN**, 2015. Disponivel em: <https://msdn.microsoft.com/en-us/library/bb397906.aspx>. Acesso em: 27 nov. 2016.

MICROSOFT CORPORATION. The MVVM Pattern, 05 julho 2016. Disponivel em: <https://msdn.microsoft.com/en-us/library/hh848246.aspx?f=255&MSPPError=-2147217396>. Acesso em: 27 nov. 2016.

MICROSOFT CORPORATION. Tour of.Net. **Microsoft Docs**, 16 novembro 2016. Disponivel em: <https://docs.microsoft.com/pt-br/dotnet/articles/standard/tour>. Acesso em: 04 fev. 2017.

MICROSOFT CORPORATION. Compilação para MSIL. **Microsoft MSDN**. Disponivel em: <https://msdn.microsoft.com/pt-br/library/c5tkafs1(v=vs.90).aspx>. Acesso em: 05 fev. 2017.

MICROSOFT CORPORATION. Licenciamento da interface do usuário do Office para desenvolvedores. **Microsoft Office DevCenter**. Disponivel em: <http://msdn.microsoft.com/officeui>. Acesso em: 14 fev. 2017.

PRESSMAN, R. S. **Engenharia de Software:** Uma abordagem profissional. 7ª. ed. [S.l.]: McGraw-Hill, 2011.

SOMMERVILLE, I. **Engenharia de Software**. 8ª Edição. ed. São Paulo: Pearson Addison-Wesley, 2007.

APÊNDICE A

Diagrama de Casos de Uso

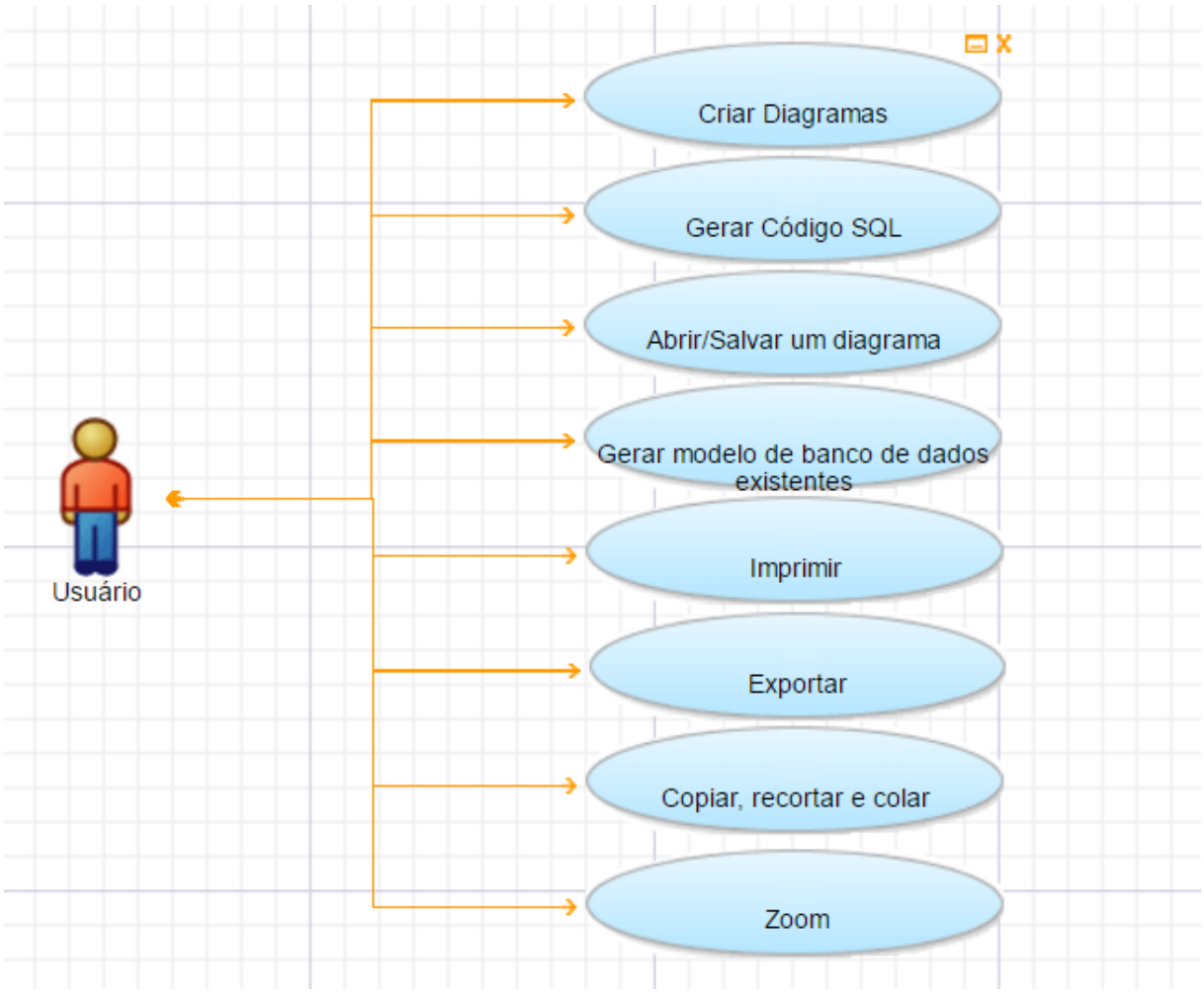


Figura 16: Caso de uso do *software*

APÊNDICE B

Formulário de Participação nos Testes de *Software*

Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas - JM

FORMULÁRIO DE PARTICIPAÇÃO EM TESTE DE SOFTWARE

1 - Objetivos Gerais

1.1 O Objetivo deste trabalho é organizar uma equipe de usuários qualificados para a realização de testes de software.

2.2 O Objetivo do teste não é avaliar a capacidade do usuário, mas os possíveis problemas, e dificuldades encontradas durante os testes do software.

2 - Usuários do Teste

2.1 Embora os testes tenham sido organizados em colaboração com o professor da disciplina de Banco de Dados I, a participação ou não por parte do aluno participação não terá impacto na nota final da disciplina.

2.2 Fica facultativo ao aluno a participação no teste do software.

2.3 O Usuário do teste se compromete a tentar resolver os problemas propostos durante o teste utilizando o software proposto.

2.4 O Usuário do teste deverá evitar interações desnecessárias com os demais participantes durante a realização dos testes, e, em nenhuma hipótese deverá discutir sobre o teste durante sua realização.

2.5 Durante o teste não é permitido ao usuário fazer perguntas sobre a utilização do software aos avaliadores.

2.6 O Usuário deverá preencher o formulário de avaliação correspondente a cada um dos testes propostos após a realização do mesmo.

2.7 As críticas do usuário deverão ser registradas no formulário entregue durante o teste.

2.8 O Usuário do teste deverá evitar a permanência na sala do teste após a finalização do mesmo.

2.9 A colaboração espontânea do usuário é fundamental para um teste bem sucedido.

3 - Avaliadores do Teste

3.1 Os avaliadores do teste são responsáveis por perceber, capturar e armazenar no formulário todas os possíveis problemas e dificuldades encontradas pelos usuários.

3.2 Os avaliadores deverão estar atentos aos detalhes durante a utilização do software por parte do usuário, como por exemplo, microexpressões faciais ou frases que indiquem possíveis problemas encontrados durante a utilização do software.

3.3 Os avaliadores do teste não deverão ajudar e nem fornecer dicas aos usuários de como utilizar o software.

Por concordar com os termos supracitados, confirmo minha presença na realização do teste de software e assino abaixo.

Nome	Matrícula

APÊNDICE C

Formulário do Avaliador

1 – Dificuldades expressas para criar um projeto:

_____

_____

_____

2 – Dificuldades expressas na modelagem de um projeto:

_____

_____

_____

3 – Dificuldades expressas para descobrir uma funcionalidade:

_____

_____

_____

4 – Dificuldades expressas para descobrir como funciona uma funcionalidade:

_____

_____

_____

5 – Dificuldades expressas por ausência de conhecimento necessário:

_____

_____

_____

6 – Quaisquer outras dificuldades expressas:

_____

_____

_____

_____

APÊNDICE D

Problemas para resolução pelos usuários

1 - Modele um diagrama Entidade-Relacionamento para um sistema de uma locadora de filmes. O banco de dados deverá armazenar os filmes (Nome, ano de lançamento, atores participantes, diretor), Clientes (Nome, CPF, RG, Endereço), Operadores (Nome, *login*, senha). O banco de dados também deverá permitir armazenar as locações do usuário com data de locação e data de entrega. Cada uma das entidades modeladas deverá ter um identificador único associadas a elas e cada um dos atributos deverão estar com o tipo de dado adequado. Após modelar o banco de dados salve o arquivo e feche o programa.

2- Reproduza o seguinte diagrama utilizando o software:

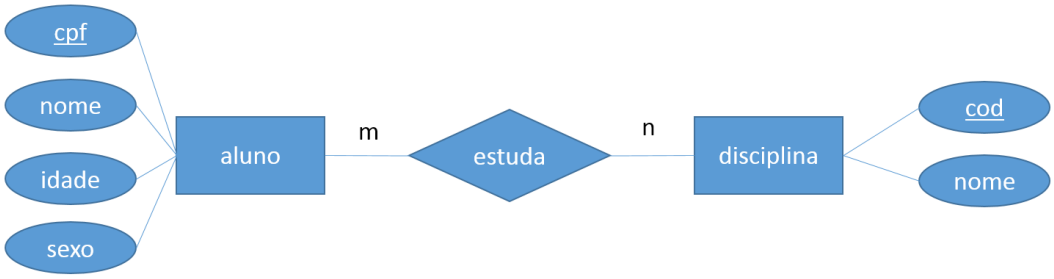


Figura 17: Diagrama ER para reprodução em testes

3- Abra o arquivo criado no teste 1. Utilizando os comandos de menu ou teclas de atalho duplique uma entidade ou relacionamento com Copiar e Colar. Aplique um tema (cores) ao diagrama. Crie um documento utilizando algum editor de textos instalado e insira uma imagem do diagrama alterado. Após inserir o diagrama, salve o arquivo de documento e o arquivo do diagrama e feche o programa.

APÊNDICE E

Formulários de avaliação aplicado (Testes 1, 2 e 3)

Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas - JM

FORMULÁRIO DE AVALIAÇÃO - TESTE 1

Aluno	Matrícula

1 - Como você classificaria sua dificuldade para criar um novo projeto?

Muito Difícil | Difícil | Razoável | Fácil | Muito Fácil

2 - Como você classificaria sua dificuldade para criar um novo arquivo?

Muito Difícil | Difícil | Razoável | Fácil | Muito Fácil

3 - Como você classificaria sua dificuldade para criar uma entidade ou relacionamento?

Muito Difícil | Difícil | Razoável | Fácil | Muito Fácil

4 - Como você classificaria o grau de dificuldade para renomear uma entidade ou relacionamento?

Muito Difícil | Difícil | Razoável | Fácil | Muito Fácil

5 - Como você classificaria sua dificuldade para adicionar atributos a uma entidade?

Muito Difícil | Difícil | Razoável | Fácil | Muito Fácil

6 - Como você classificaria sua dificuldade para modificar o tipo de um atributo?

Muito Difícil | Difícil | Razoável | Fácil | Muito Fácil

7 - Como você classificaria sua dificuldade para conectar uma entidade a um relacionamento?

Muito Difícil | Difícil | Razoável | Fácil | Muito Fácil

8 - Como você classificaria o grau de difiucldade para modificar a cardinalidade de um relacionamento?

Muito Difícil | Difícil | Razoável | Fácil | Muito Fácil

9 - Como você classificaria o grau de difucldade para salvar um arquivo?

Muito Difícil | Difícil | Razoável | Fácil | Muito Fácil

10 - Críticas ou sugestões

Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas - JM

Aluno	Matrícula

FORMULÁRIO DE AVALIAÇÃO - TESTE 2

1 - Como você classificaria sua dificuldade para reproduzir o problema com exatidão?

Muito Difícil | Difícil | Razoável | Fácil | Muito Fácil

2 - Como você classificaria sua dificuldade para encontrar as opções necessárias para a tarefa?

Muito Difícil | Difícil | Razoável | Fácil | Muito Fácil

3 - Como você classificaria sua dificuldade para gerar o código SQL para o diagrama?

Muito Difícil | Difícil | Razoável | Fácil | Muito Fácil

4 - Como você classificaria o grau de dificuldade para renomear uma entidade ou relacionamento?

Muito Difícil | Difícil | Razoável | Fácil | Muito Fácil

5 - Como você classificaria o tempo de resposta para geração de código SQL?

Muito Ruim | Ruim | Razoável | Bom | Muito Bom

6 - Como você classificaria a forma com que o código SQL é apresentado?

Muito Ruim | Ruim | Razoável | Bom | Muito Bom

7 - Como você classificaria sua dificuldade para corrigir algum erro cometido?

Muito Difícil | Difícil | Razoável | Fácil | Muito Fácil

10 - Críticas ou sugestões

Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas - JM

Aluno	Matrícula

FORMULÁRIO DE AVALIAÇÃO - TESTE 3

1 - Como você classificaria sua dificuldade para abrir um arquivo?

Muito Difícil | Difícil | Razoável | Fácil | Muito Fácil

2 - Como você classificaria sua dificuldade para duplicar uma entidade ou relacionamento?

Muito Difícil | Difícil | Razoável | Fácil | Muito Fácil

3 - Como você classificaria sua dificuldade para aplicar um tema (cores) ao diagrama?

Muito Difícil | Difícil | Razoável | Fácil | Muito Fácil

4 - Como você classificaria a qualidade da interface gráfica?

Muito Ruim | Ruim | Razoável | Bom | Muito Bom

5 - Como você classificaria a facilidade para descobrir uma funcionalidade?

Muito Ruim | Ruim | Razoável | Bom | Muito Bom

6 - Como você classificaria o tamanho dos ícones da interface gráfica?

Muito Ruim | Ruim | Razoável | Bom | Muito Bom

7 - Como você classificaria sua dificuldade para navegar entre as abas do sistema?

Muito Difícil | Difícil | Razoável | Fácil | Muito Fácil

8 - Como você classificaria a interface gráfica no geral?

Muito Difícil | Difícil | Razoável | Fácil | Muito Fácil

8 - Como você classificaria o software no geral?

Muito Difícil | Difícil | Razoável | Fácil | Muito Fácil

9 - Você utilizaria o software em um de seus projetos futuros?

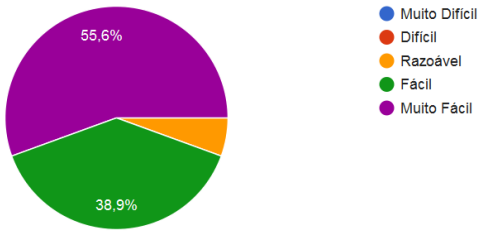
( )Sim ( ) Não

10 - Críticas ou Sugestões

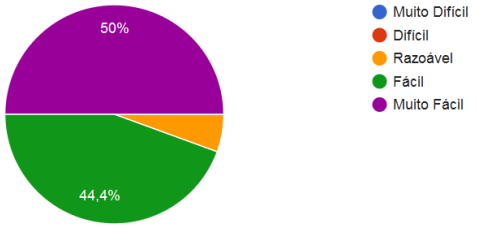
APÊNDICE F

Resultados do formulário 1 aplicado

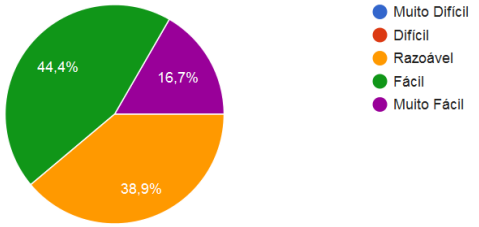
Como você classificaria sua dificuldade para criar um novo projeto?
(18 respostas)



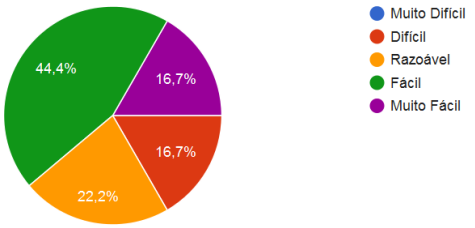
Como você classificaria sua dificuldade para criar um novo arquivo?
(18 respostas)



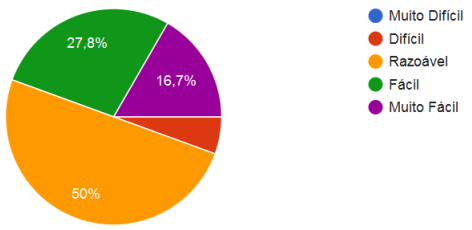
Como você classificaria sua dificuldade para criar uma entidade ou relacionamento?
(18 respostas)



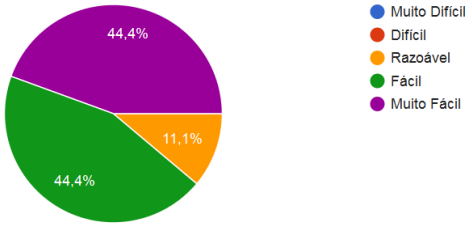
Como você classificaria sua dificuldade para renomear uma entidade ou relacionamento?
(18 respostas)



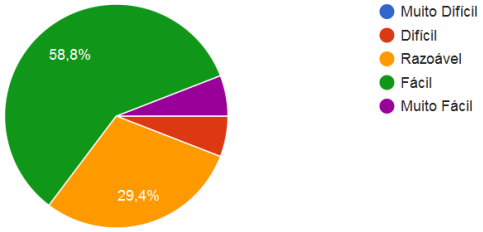
Como você classificaria sua dificuldade para adicionar atributos a uma entidade?
(18 respostas)



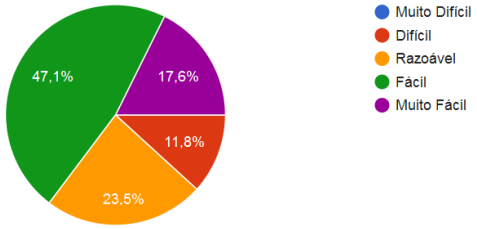
Como você classificaria sua dificuldade para modificar o tipo de um atributo?
(18 respostas)



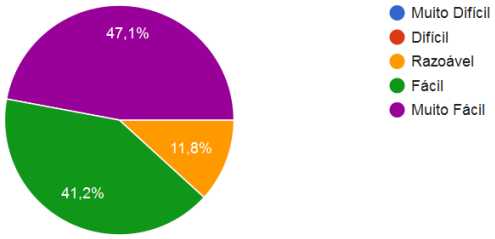
Como você classificaria sua dificuldade para conectar uma entidade a um relacionamento?
(17 respostas)



Como você classificaria sua dificuldade para modificar a cardinalidade de um relacionamento?
(17 respostas)



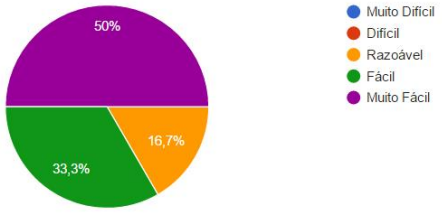
Como você classificaria sua dificuldade para salvar um arquivo? (17 respostas)



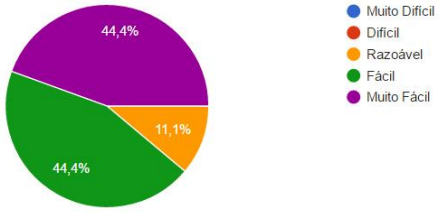
APÊNDICE G

Resultados do formulário 2 aplicado

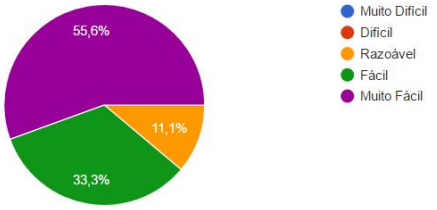
Como você classificaria sua dificuldade para reproduzir o problema com exatidão?
(18 respostas)



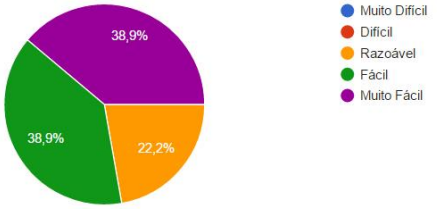
Como você classificaria sua dificuldade para encontrar as opções necessárias para a tarefa?
(18 respostas)



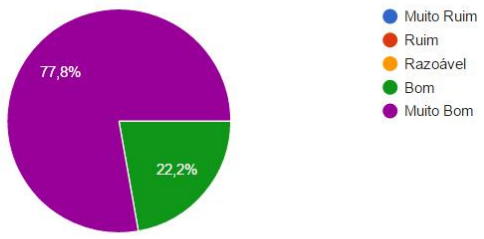
Como você classificaria sua dificuldade para gerar o código SQL para o diagrama?
(18 respostas)



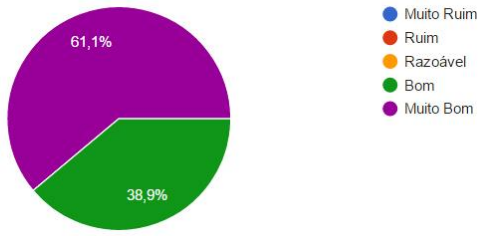
Como você classificaria sua dificuldade para renomear uma entidade ou relacionamento?
(18 respostas)



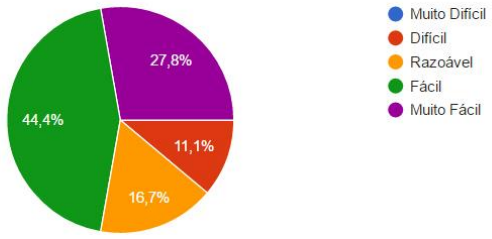
Como você classificaria o tempo de resposta para a geração do código SQL?
(18 respostas)



Como você classificaria a forma com que o código SQL é apresentado?
(18 respostas)



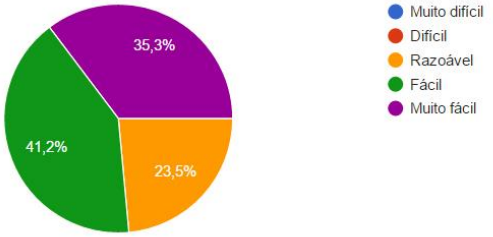
Como você classificaria sua dificuldade para corrigir algum erro cometido?
(18 respostas)



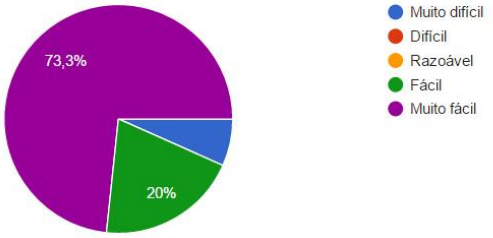
APÊNDICE H

Resultados do formulário 3 aplicado

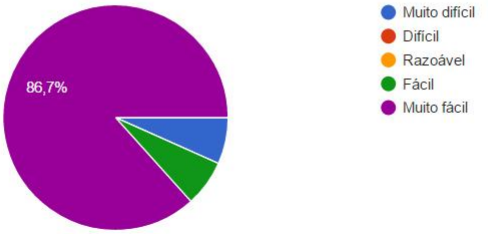
Como você classificaria sua dificuldade para abrir um arquivo? (17 respostas)



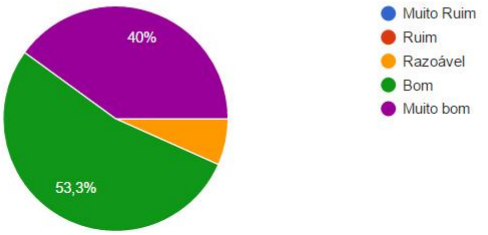
Como você classificaria sua dificuldade para duplicar uma entidade ou relacionamento?
(15 respostas)



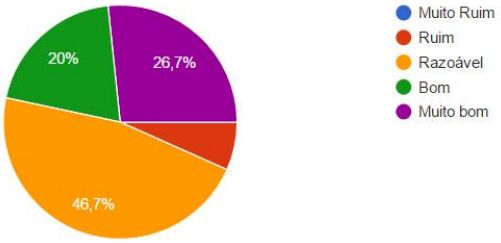
Como você classificaria sua dificuldade para aplicar um tema (cores) ao diagrama?
(15 respostas)



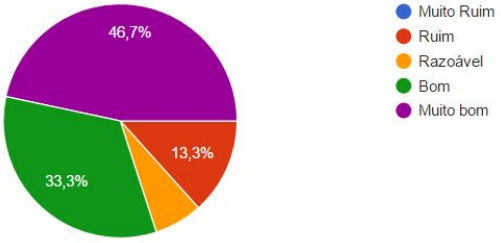
Como você classificaria a qualidade da interface gráfica? (15 respostas)



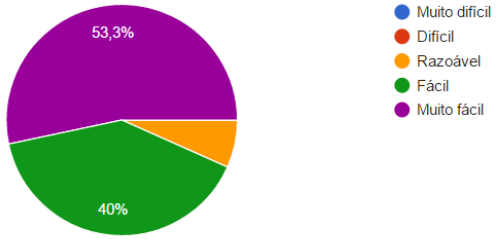
Como você classificaria a facilidade para descobrir uma funcionalidade?
(15 respostas)



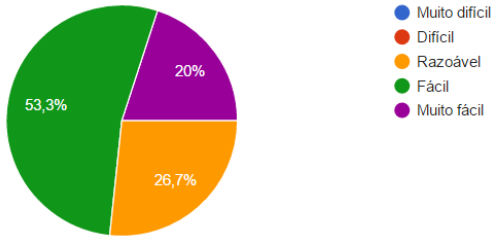
Como você classificaria o tamanho dos ícones da interface gráfica?
(15 respostas)



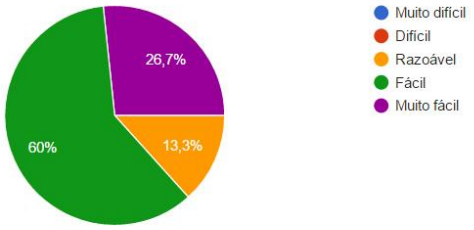
Como você classificaria sua dificuldade para navegar entre as abas do sistema?
(15 respostas)



Como você classificaria a interface gráfica no geral? (15 respostas)



Como você classificaria o software no geral? (15 respostas)



Você utilizaria o software em um de seus projetos futuros? (17 respostas)