



# Bug Tracking

## Implementando Bug Tracking em uma aplicação apoiada por Controle de Mudanças



### Felipe La Rocca Teixeira

(felipe.lr@pjf.mg.gov.br)

é Bacharel em Sistemas de Informação pelo Centro de Ensino Superior de Juiz de Fora (CES/JF) e Analista de Sistemas da Prefeitura de Juiz de Fora.



### Marco Antônio Pereira Araújo

(maraujo@cesjf.br)

é professor do Curso de Bacharelado em Sistemas de Informação do Centro de Ensino Superior de Juiz de Fora (CES/JF), Doutorando e Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ, Especialista em Métodos Estatísticos Computacionais e Bacharel em Informática pela UFJF, Analista de Sistemas da Prefeitura de Juiz de Fora.

Atualmente com o rápido avanço da tecnologia, desenvolver e manter sistemas de informação tornou-se algo difícil e complexo. Toda essa nova tecnologia também gera um aumento de novas solicitações e expectativas do usuário final e, para que possamos construir softwares de qualidade, é necessário não somente uma equipe técnica competente, mas também uma comunicação efetiva entre equipe de desenvolvimento e usuário final.

Ao longo do ciclo de vida de um software, modificações são inevitáveis, podendo ocorrer mudanças nas necessidades dos usuários, alterações do ambiente onde o sistema será utilizado, correções de defeitos, melhorias das funcionalidades do sistema.

Essas modificações afetam todos os tipos de artefatos como requisitos do sistema, documentos de análise e projeto, código-fonte e testes. A Gerência de Configuração é um conjunto de ativida-

des projetadas para controlar mudanças inerentes ao desenvolvimento, mantendo a estabilidade na evolução do software, desempenhando um papel importante durante o desenvolvimento do projeto, tais como:

- Identificar os itens de configuração de acordo com as funcionalidades que deverão desempenhar;
- Documentar os itens ao longo do processo de desenvolvimento, gerando interação entre eles;
- Controlar as modificações ocorridas nos itens;
- Auditar modificações, garantindo a confiabilidade do produto.

Não é objetivo da gerência de configuração evitar modificações, mas permitir que elas ocorram de maneira controlada. Neste artigo será discutida uma de suas atividades principais, que é o controle de mudança.

Esse controle pode ser feito através do uso de ferramentas conhecidas como

*Bug Tracking*, permitindo assim ao desenvolvedor rastrear essas mudanças e identificar o impacto no projeto como um todo.

## Controle de mudanças

É uma coleção de procedimentos documentados e formais que definem como as mudanças no software serão monitoradas e avaliadas. Oferece processos para identificar, analisar, rastrear e controlar mudanças. O uso de ferramentas possibilita a melhoria dos processos utilizados no desenvolvimento e manutenção de produtos de software.

As ferramentas de *Bug Tracking* estão prontas para auxiliar desenvolvedores no controle de mudanças, permitindo também que usuários façam o acompanhamento completo dos pedidos de alterações sobre erros encontrados no sistema. No mercado existem várias ferramentas *open source* de *Bug Tracking* disponíveis como *Bugzilla* e *Trac*.

Principais funcionalidades que uma ferramenta de Controle de Mudanças deve possuir:

- Acompanhamento de todo o ciclo de vida do pedido de mudança;
- Anexar arquivos ao pedido para facilitar o entendimento do problema ou complementar a especificação;
- Configuração do fluxo de trabalho, definindo os estados pelo qual um pedido passa durante o seu ciclo de vida;
- Notificações de acompanhamento, mantendo informados todos os envolvidos com um pedido de mudança sobre alterações recebidas durante o ciclo de vida;
- Possibilitar o uso de campos personalizados, caso seja necessário acrescentar mais informações para atender a uma necessidade específica do projeto;
- Rastreamento de mudanças identificando todas as suas dependências;
- Relatórios personalizados para consulta de toda a evolução dos pedidos de mudança.

## Trac

A ferramenta *Trac* é uma das mais completas. Baseada na Web é implementada como um CGI ou como um programa independente. Foi desenvolvida na linguagem de programação *Python*, e

possui alguns diferenciais como acompanhamento da evolução do projeto, integração com a ferramenta *Subversion* e outras ferramentas de controle de versões, arquitetura de *plugins*, além de suporte para os bancos de dados *SQLite*, *PostgreSQL* e *MySQL*.

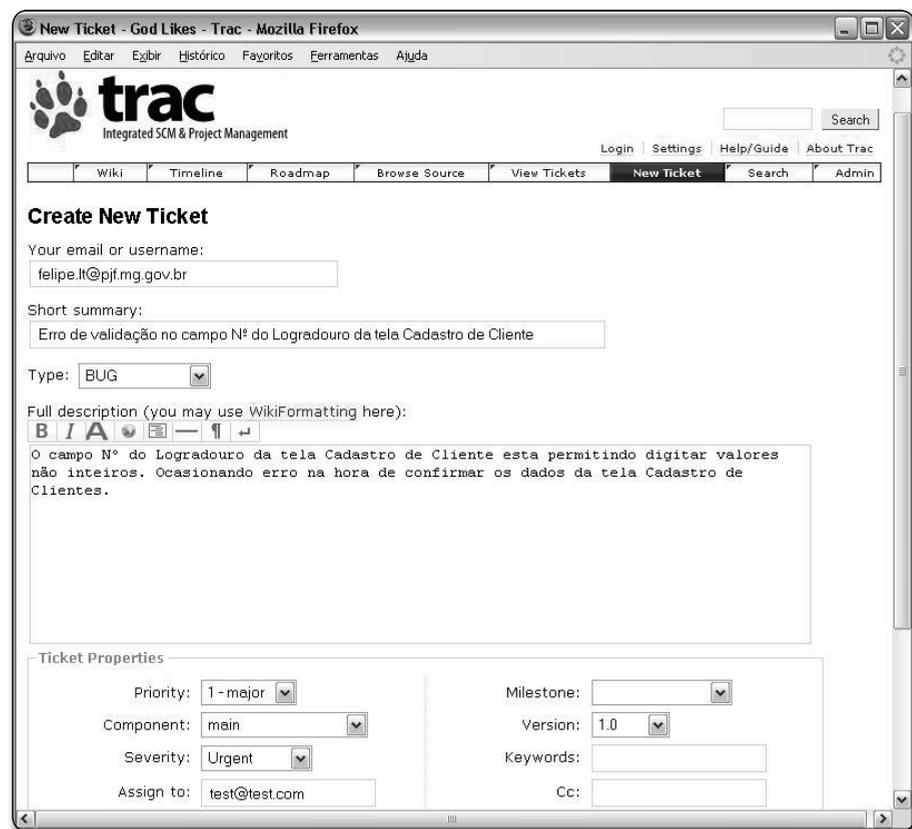
No endereço [www.hosted-projects.com/trac/TracDemo/Demo](http://www.hosted-projects.com/trac/TracDemo/Demo) podemos testar a ferramenta através de um demo. Para reportar um erro, clique no menu *New Ticket*. Na **Figura 1** vemos como a estrutura do *Trac* é bem simples, utilizando os campos:

- *Your email or username*: responsável em reportar um problema ou uma nova necessidade do usuário do software;
- *Short summary*: definição objetiva do problema encontrado ou nova necessidade;
- *Type*: tipo do que está sendo reportado, um problema ou uma nova necessidade;
- *Full description*: descrição completa do problema ou nova necessidade;
- *Priority*: usado para marcar a urgência da resolução do que está sendo reportado;

- *Component*: subsistema ou módulo de um produto;
- *Severity*: MOSTRA o quanto é relevante o que está sendo reportado;
- *Version*: indica a versão do software que ocorreu o problema ou a nova necessidade;
- *Assign to*: responsável pela resolução do problema ou implementação da nova necessidade;
- *Cc*: lista de pessoas que podem ser informadas sobre o que está sendo reportado;
- *Keywords*: palavras-chave para categorizar o que está sendo reportado.

Uma de suas características mais interessantes é a formatação *Wiki*. Um *Wiki* é utilizado para identificar um tipo específico de coleção de documentos em hipertexto ou o conteúdo de um documento que pode ficar disponível a qualquer momento através de um navegador.

Todas as informações adicionais e modificações feitas após a criação do *ticket* são mantidas, gerando um histórico da evolução do erro. Dessa forma, as mudanças que foram solicitadas e as



**Figura 1.** Visualização de um Ticket na ferramenta Trac

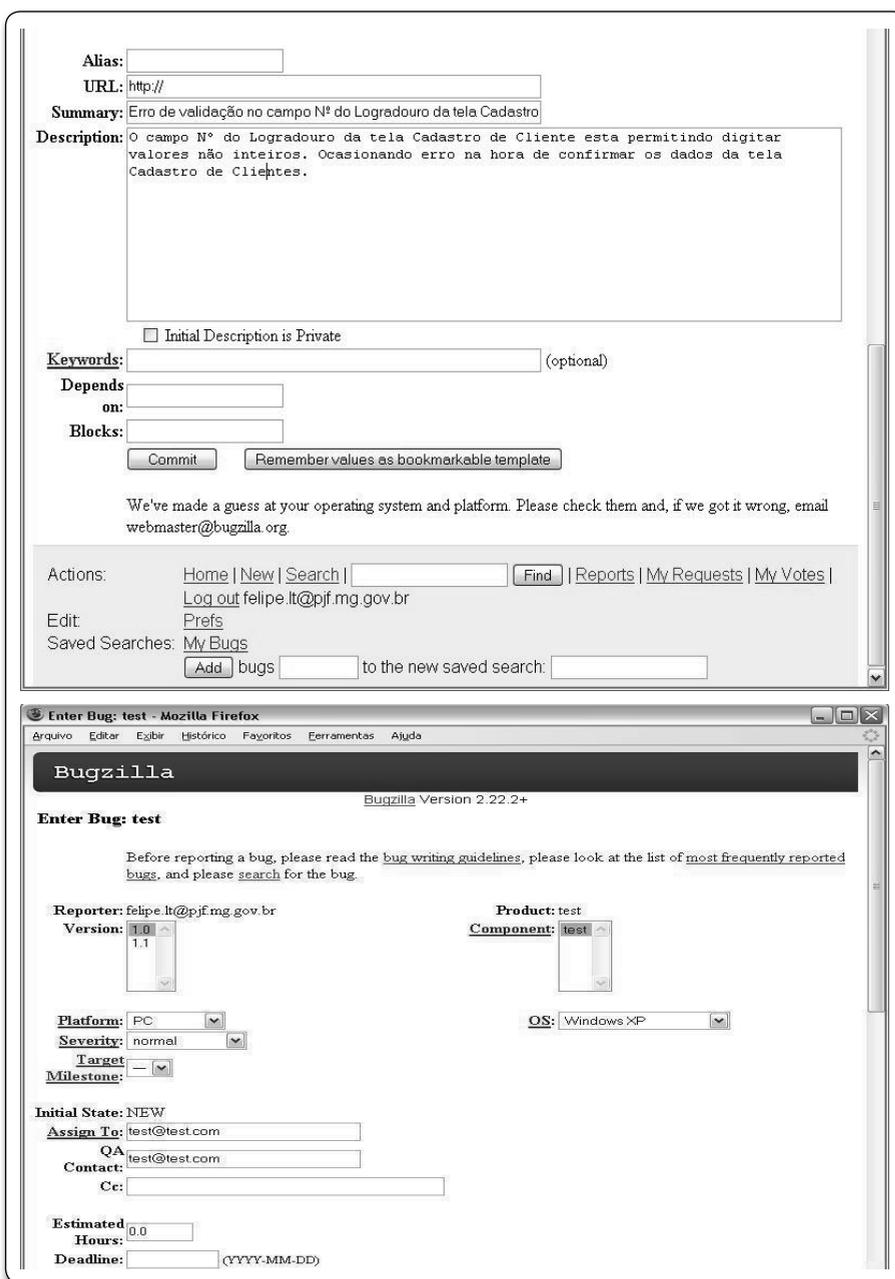


Figura 2. Visualização do Enter Bug

correções feitas no software a partir delas são rastreadas pelos *links* criados através da formatação *Wiki*.

## Bugzilla

A ferramenta *Bugzilla* é uma das mais utilizadas entre os desenvolvedores, porém é de mais difícil instalação e, na instalação *default*, ainda precisa de uma grande quantidade de informações para descrever um problema.

Ela também é baseada na Web e dá suporte ao desenvolvimento do projeto *Mozilla*, permitindo o rastreamento de alterações e servindo também como plataforma para pedidos de novas solicitações.

No endereço [landfill.bugzilla.org/bugzilla-2.22-branch](http://landfill.bugzilla.org/bugzilla-2.22-branch) podemos testar a ferramenta através de um demo. Primeiramente, para reportar um erro é necessário criar uma conta no *Bugzilla*, depois clicar em *Enter a new bug>test product*, conforme **Figura 2**.

Além de possuir alguns campos com as mesmas características da ferramenta *Trac*, também se pode destacar:

- *Reporter*: responsável em reportar um problema ou nova necessidade do usuário do software;
- *Product*: qual software o problema ou nova necessidade se refere;
- *Platform e OS*: indica o ambiente onde o problema ou nova necessidade ocorreu;
- *Initial State*: estado que um problema ou nova solicitação se encontra, por exemplo, informa se o problema é novo;
- *Estimated Hours*: estimativa do esforço gasto para correção do problema ou implementação da nova necessidade;



- **Deadline:** previsão para o concerto do problema ou implementação da nova necessidade;
- **URL:** uma URL associada ao que está sendo reportado;
- **Depends on:** informa dependências entre problemas ou novas necessidades.

## Bug Tracking na prática

Para demonstrar na prática a utilização de *Bug Tracking* em um sistema, este artigo vai apresentar a criação de uma aplicação de exemplo contendo uma janela simples de Cadastro de Cliente, além de uma opção no menu da aplicação para que o usuário possa reportar erros ou novas necessidades diretamente para a equipe de desenvolvimento.

Além disso, quando um erro não tratado na aplicação ocorrer, esse também será capturado e enviado automaticamente para os desenvolvedores da aplicação. Primeiramente, temos que criar um banco de dados InterBase/Firebird com o nome de “BUGTRACKING.GDB” contendo as tabelas CLIENTE e ERRO, mostradas na *script* da **Listagem 1**, além dos *Generators* e *Stored Procedures*..

A seguir, vamos criar uma nova aplicação no Delphi, salvar a unit com o nome de “ufmPadrao.pas” e o projeto como “BugTracking.dpr”. Essa unit será definida como o formulário padrão da aplicação, e os novos formulários serão criados a partir dela, facilitando o processo de desenvolvimento.

Para isso, adicione o formulário ao repositório do Delphi, clicando com o botão direito e escolhendo o item *Add to Repository* configurando-o como apresentado na **Figura 3**.

Dando continuidade ao projeto, para acessar o banco de dados criado, devemos criar um Data Module em *File>New>Data Module* e salvar com o nome de “udtmdlPrincipal.pas”. Adicione os componentes *SQLConnection*, *SQLStoredProc*, *SQLDataSet*, *DataSetProvider* e *ClientDataSet* (**Figura 4**). Devemos então definir seus nomes e propriedades conforme apresentado na **Tabela 1**, respectivamente.

## Capturando erros da aplicação

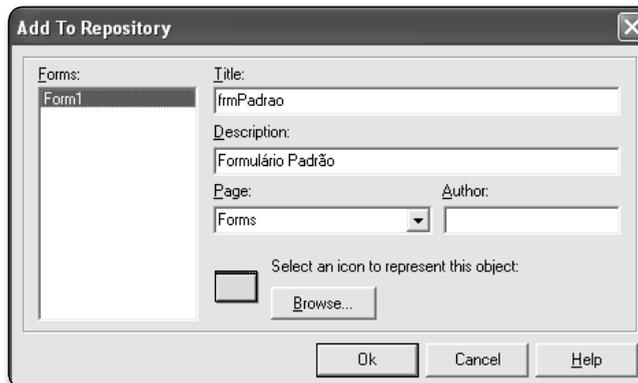
Com o Data Module salvo ao projeto, podemos implementar a função “CapturaErro” na seção *private* em *ufmPa-*

**Listagem 1.** Objetos do banco de dados

```
CREATE TABLE CLIENTE (
  ID_CLIENTE INTEGER NOT NULL,
  NOME VARCHAR(70),
  LOGRADOURO VARCHAR(50),
  NUMERO INTEGER,
  COMPLEMENTO VARCHAR(10),
  BAIRRO VARCHAR(30));
ALTER TABLE CLIENTE ADD CONSTRAINT PK_CLIENTE
PRIMARY KEY (ID_CLIENTE);
CREATE TABLE ERRO (
  ID_ERRO INTEGER NOT NULL,
  SUMARIO VARCHAR(50),
  DESCRICAO BLOB SUB_TYPE 0 SEGMENT SIZE 500,
  USUARIO VARCHAR(30),
  DATA TIMESTAMP,
  TIPO INTEGER,
  CONTROLE VARCHAR(30),
  FORMULARIO VARCHAR(30),
  SEVERIDADE INTEGER,
  STATUS INTEGER,
  VERSAO VARCHAR(10));
ALTER TABLE ERRO ADD CONSTRAINT PK_ERRO
PRIMARY KEY (ID_ERRO);
CREATE GENERATOR GEN_CLIENTE_ID;
CREATE GENERATOR GEN_ERRO_ID;
CREATE PROCEDURE SP_GEN_CLIENTE_ID
RETURNS (
  ID INTEGER)
AS
BEGIN
  ID = GEN_ID(GEN_CLIENTE_ID, 1);
  SUSPEND;
END
CREATE PROCEDURE SP_GEN_ERRO_ID
RETURNS (
  ID INTEGER)
AS
BEGIN
  ID = GEN_ID(GEN_ERRO_ID, 1);
  SUSPEND;
END
```

**Listagem 2.** Função para Capturar erros não tratados pelo sistema

```
uses utmdlPrincipal;
...
procedure TfrmPadrao.CapturarErro(Sender: TObject;
  Excpn: Exception);
begin
  dtmdlPrincipal.cIntdtstPrincipal.Close;
  dtmdlPrincipal.sqldtstPrincipal.CommandText := 'select * from ERRO';
  dtmdlPrincipal.cIntdtstPrincipal.Open;
  dtmdlPrincipal.cIntdtstPrincipal.Insert;
  dtmdlPrincipal.sqlspPrincipal.StoredProcName := 'SP_GEN_ERRO_ID';
  dtmdlPrincipal.sqlspPrincipal.ExecProc;
  dtmdlPrincipal.cIntdtstPrincipal.FieldByName(
    'ID_ERRO').AsInteger := dtmdlPrincipal.sqlspPrincipal.Params[0].AsInteger;
  dtmdlPrincipal.cIntdtstPrincipal.FieldByName('USUARIO').AsString := 'admin';
  dtmdlPrincipal.cIntdtstPrincipal.FieldByName('DATA').AsDateTime := Now;
  dtmdlPrincipal.cIntdtstPrincipal.FieldByName('TIPO').AsInteger := 0;
  dtmdlPrincipal.cIntdtstPrincipal.FieldByName(
    'SUMARIO').AsString := 'Erro do Sistema em ' + Screen.ActiveForm.Caption;
  dtmdlPrincipal.cIntdtstPrincipal.FieldByName('DESCRICAO').AsString := Excpn.message;
  dtmdlPrincipal.cIntdtstPrincipal.FieldByName('FORMULARIO').AsString := Screen.ActiveForm.Name;
  dtmdlPrincipal.cIntdtstPrincipal.FieldByName(
    'CONTROLE').AsString := Screen.ActiveControl.Name;
  dtmdlPrincipal.cIntdtstPrincipal.FieldByName('SEVERIDADE').AsInteger := 0;
  dtmdlPrincipal.cIntdtstPrincipal.FieldByName('STATUS').AsInteger := 0;
  dtmdlPrincipal.cIntdtstPrincipal.FieldByName('VERSAO').AsString := 'v1.0';
  dtmdlPrincipal.cIntdtstPrincipal.ApplyUpdates(0);
  MessageDlg(Excpn.message + ' Este erro será gravado e poderá ser enviado '+
    'posteriormente para a equipe de Desenvolvimento.', mtError, [mbOK], 0);
end;
```



**Figura 3.** Visualização da janela Add To Repository

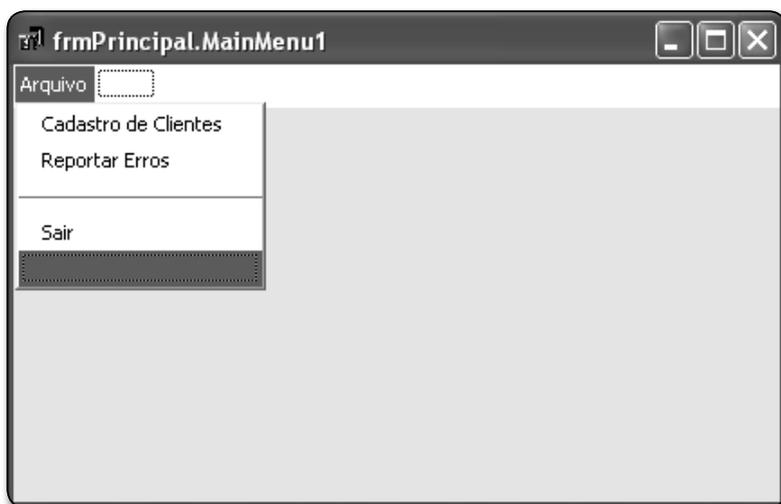


Nome	Propriedade	Valor
sqlcnctnConexao	Params	Configurar o caminho do banco
sqlspPrincipal	SQLConnection	sqlcnctnConexao
sqldtstPrincipal	SQLConnection	sqlcnctnConexao
dtstprvdrPrincipal	DataSet	sqldtstPrincipal
clntdtstPrincipal	ProviderName	dtstprvdrPrincipal

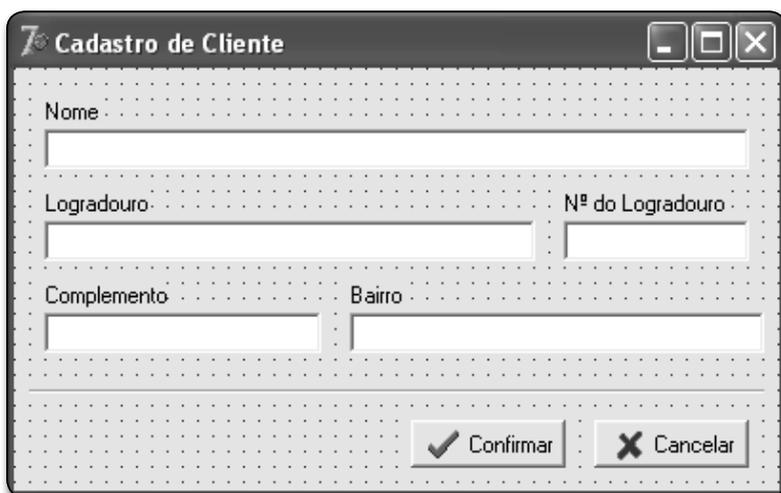
**Tabela 1.** Configurações dos componentes de acesso ao banco de dados

Campo	Valor	Corresponde
ID_ERRO	sqlspPrincipal.Params[0].AsInteger	Executar <i>procedure</i> auto-incremento
USUARIO	admin	Valor <i>default</i> do usuário do sistema
DATA	Now	Momento que ocorreu o erro
TIPO	0	0=Erro / 1=Nova Solicitação
SUMARIO	Screen.ActiveForm.Caption	Nome da janela que ocorreu o erro
DESCRICAÇÃO	Exception.message	Mensagem de erro que foi exibida
FORMULARIO	Screen.ActiveForm.Name	Formulário que ocorreu o erro
CONTROLE	Screen.ActiveControl.Name	Componente que gerou o erro
SEVERIDADE	0	0=Alta / 1=Média / 2=Baixa
STATUS	0	0=Em Aberto / 1=Corrigido / 3=Rejeitado
VERSAO	V1.0	Valor <i>default</i> da versão do software

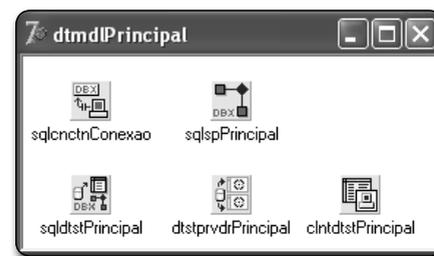
**Tabela 2.** Valores manipulados pela função CapturarErro



**Figura 5.** Visualização do menu da janela principal da aplicação



**Figura 6.** Visualização da tela Cadastro de Cliente



**Figura 4.** Visualização do Data Module

drao criado anteriormente, conforme a **Listagem 2**.

Essa função capturará o máximo de informações possíveis geradas por um erro inesperado no sistema e gravará na tabela ERRO. Dessa forma, um erro ocorrido poderá ser enviado posteriormente para a equipe de desenvolvimento do software.

Na **Tabela 2** são apresentadas algumas características importantes da *Captura-Erro*.

Para finalizar a codificação desse formulário, adicione no *OnCreate* do mesmo, o seguinte código:

```
Application.OnException := CapturarErro;
```

Com o formulário padrão adicionado ao repositório do Delphi, podemos criar uma tela principal para o sistema, em *File>New>Other*. Selecione a aba *Form*, escolha *frmPadrao* e marque o item *Inherit*, para que o novo formulário herde todas as características do formulário padrão, como a *CapturaErro*. Adicione um *MainMenu* com os itens da **Figura 5** e salve a unit com o nome de "ufmPrincipal.pas".

Também devemos criar uma tela simples de cadastro, seguindo os mesmos passos da tela principal, salvando a unit como "ufmCadastroCliente.pas" e configurar um *layout* semelhante a **Figura 6**.

Na *ufmCadastroCliente* no evento *OnClick* do *Confirmar*, digite o código apresentado na **Listagem 3**, para gravação dos dados (atente para o nome dos componentes de tela). Neste exemplo, veja que não foi implementado nenhum tratamento dos valores informados no campo N° do Logradouro.

Em nosso banco de dados o campo NUMERO é do tipo INTEGER, podendo ocorrer de um usuário digitar um valor não numérico nesse campo, gerando erro na hora de gravar os dados.

## Listando os erros capturados

Para visualizar os erros cadastrados devemos criar um novo formulário usando a mesma técnica anterior, e salvar a *unit* como "ufmListaErros.pas". Então adicione os componentes *SQLStoredProc*, *SQLDataSet*, *DataSetProvider*, *ClientDataSet*, *DataSouce* e um *DBGrid* para exibir a lista com os erros e novas solicitações gerados no sistema, conforme a **Figura 7**.

A configuração dos componentes é parecida com a do Data Module mas com algumas diferenças, começando pelos nomes que terão os finais "Grade" como, por exemplo, o nome do *DataSouce* será *dtsrcGrade*.

Também devemos deixar a propriedade *SQLConnection* do *sqldstGrade* vazia, pois o valor será atribuído em tempo de execução pelo código da **Listagem 4**, através do evento *OnCreate* do formulário, fazendo a conexão com o banco de dados.

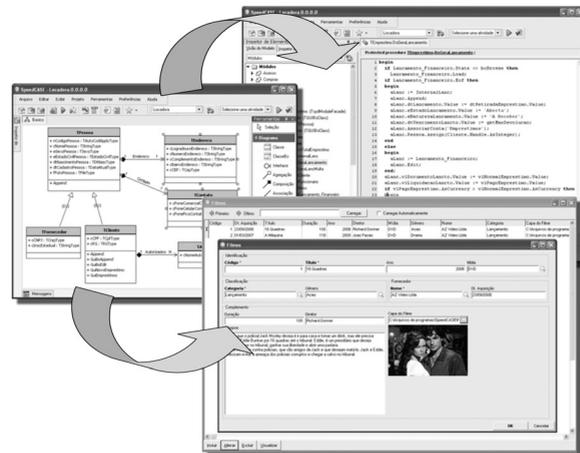
**Listagem 3.** Evento *OnClick* do Confirmar da janela Cadastro de Cliente

```
uses utmdlPrincipal;
...
procedure TfrmCadastroCliente.btnConfirmarClick(Sender: TObject);
begin
  inherited;
  dtmdlPrincipal.cIntdtstPrincipal.Close;
  dtmdlPrincipal.sqldstPrincipal.CommandText := 'select * from CLIENTE';
  dtmdlPrincipal.cIntdtstPrincipal.Open;
  dtmdlPrincipal.cIntdtstPrincipal.Insert;
  dtmdlPrincipal.sqlspPrincipal.StoredProcName := 'SP_GEN_CLIENTE_ID';
  dtmdlPrincipal.sqlspPrincipal.ExecProc;
  dtmdlPrincipal.cIntdtstPrincipal.FieldName(
    'ID_CLIENTE').AsInteger := dtmdlPrincipal.sqlspPrincipal.Params[0].AsInteger;
  dtmdlPrincipal.cIntdtstPrincipal.FieldName('NOME').AsString := edtNome.Text;
  dtmdlPrincipal.cIntdtstPrincipal.FieldName('LOGRADOURO').AsString := edtLogradouro.Text;
  dtmdlPrincipal.cIntdtstPrincipal.FieldName('NUMERO').AsString := edtNumero.Text;
  dtmdlPrincipal.cIntdtstPrincipal.FieldName('COMPLEMENTO').AsString := edtComplemento.Text;
  dtmdlPrincipal.cIntdtstPrincipal.FieldName('BAIRRO').AsString := edtBairro.Text;
  dtmdlPrincipal.cIntdtstPrincipal.ApplyUpdates(0);
  Application.MessageBox('Os dados foram incluídos.', 'Bug Report', MB_IconInformation);
end;
```

**Listagem 4.** Código do *OnCreate* do formulário de exibição de erros

```
uses utmdlPrincipal;
...
dtmdlPrincipal := TdtmdlPrincipal(
  application.FindComponent('dtmdlPrincipal') as TDataModule);
sqldstGrade.SQLConnection := dtmdlPrincipal.sqlcnctnConexao;
cIntdtstGrade.Close;
sqldstGrade.CommandText := 'select * from ERRO';
cIntdtstGrade.Open;
```

# Reduza o tempo de desenvolvimento em até 80%



Ambiente de Análise e Implementação Integrados.  
Interface Gráfica e Banco de Dados gerados a partir do modelo.

Ferramenta de produtividade para desenvolvimento de software

Conheça a versão Free!

Persistência com:  
Interbase  
Firebird  
MySQL  
MS SQL Server

Compatível com:  
Turbo Delphi  
Delphi  
Kylix

Faça do tempo seu diferencial

Desenvolva para:  
Windows  
Linux

TecnoSpeed TI - Fone: (44) 3028-3749 - [www.speedcase.com.br](http://www.speedcase.com.br)

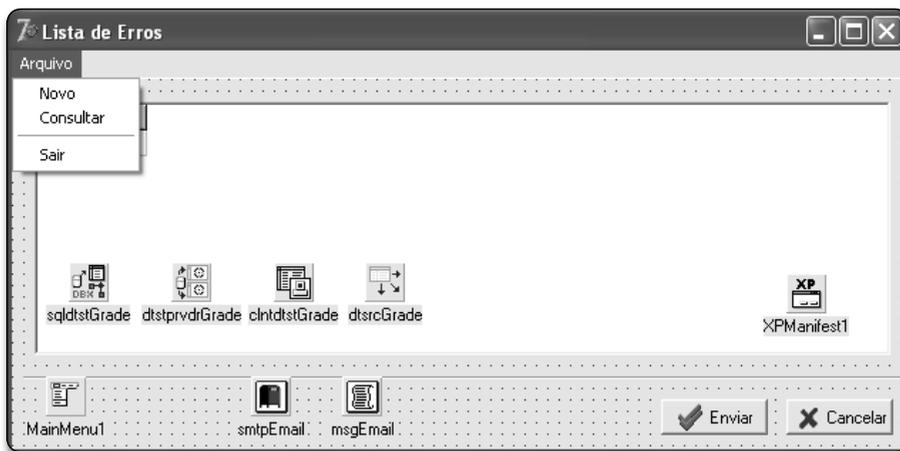


Figura 7. Visualização da janela Lista de Erros

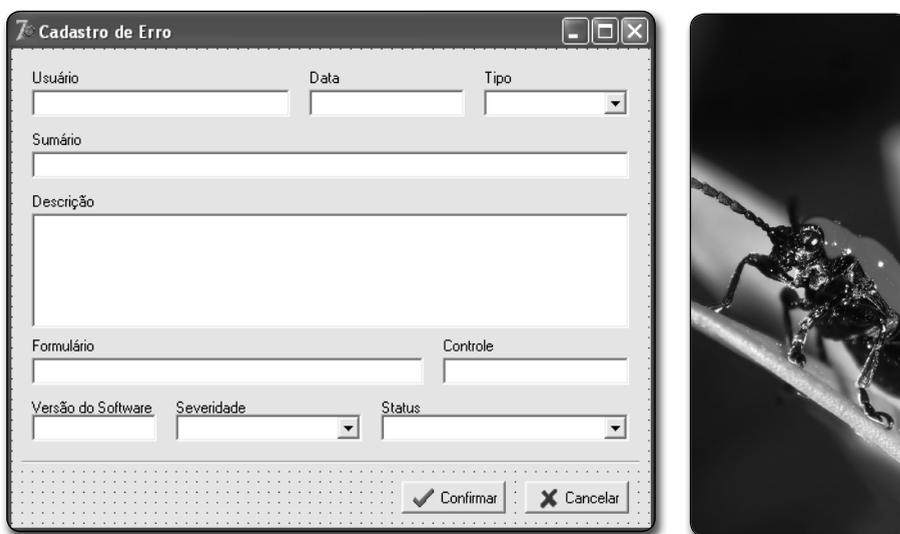


Figura 8. Visualização da janela Cadastro de Erro

#### Listagem 5. Evento OnClick do Confirmar, Tela Cadastro de Erro

```

uses utmdlPrincipal;
...
procedure TfrmCadastroErro.btnConfirmarClick(Sender: TObject);
begin
  inherited;
  dtmdlPrincipal.clnTdtstPrincipal.Close;
  dtmdlPrincipal.sqlTdtstPrincipal.CommandText := 'select * from ERRO';
  dtmdlPrincipal.clnTdtstPrincipal.Open;
  dtmdlPrincipal.clnTdtstPrincipal.Insert;
  dtmdlPrincipal.sqlspPrincipal.StoredProcName := 'SP_GEN_ERRO_ID';
  dtmdlPrincipal.sqlspPrincipal.ExecProc;
  dtmdlPrincipal.clnTdtstPrincipal.FieldName(
    'ID_ERRO').AsInteger := dtmdlPrincipal.sqlspPrincipal.Params[0].AsInteger;
  dtmdlPrincipal.clnTdtstPrincipal.FieldName('USUARIO').AsString := edtUsuario.Text;
  dtmdlPrincipal.clnTdtstPrincipal.FieldName('FORMULARIO').AsString := edtFormulario.Text;
  if cmbxTipo.ItemIndex <> -1 then
  begin
    dtmdlPrincipal.clnTdtstPrincipal.FieldName('TIPO').AsInteger := cmbxTipo.ItemIndex;
  end;
  dtmdlPrincipal.clnTdtstPrincipal.FieldName('SUMARIO').AsString := edtSumario.Text;
  dtmdlPrincipal.clnTdtstPrincipal.FieldName('DESCRICAO').AsString := mDescricao.Lines.Text;
  dtmdlPrincipal.clnTdtstPrincipal.FieldName('FORMULARIO').AsString := edtFormulario.Text;
  dtmdlPrincipal.clnTdtstPrincipal.FieldName('CONTROLE').AsString := edtControle.Text;
  if cmbxSeveridade.ItemIndex <> -1 then
  begin
    dtmdlPrincipal.clnTdtstPrincipal.FieldName('SEVERIDADE').AsInteger := cmbxSeveridade.ItemIndex;
  end;
  if cmbxStatus.ItemIndex <> -1 then
  begin
    dtmdlPrincipal.clnTdtstPrincipal.FieldName('STATUS').AsInteger := cmbxStatus.ItemIndex;
  end;
  dtmdlPrincipal.clnTdtstPrincipal.FieldName('VERSAO').AsString := edtVersao.Text;
  dtmdlPrincipal.clnTdtstPrincipal.ApplyUpdates(0);
  Application.MessageBox('Os dados foram incluídos.', 'Bug Tracking', MB_IconInformation);
end;

```

No código anterior, também indicamos o valor do *CommandText* do *sqlTdtstGrade*. Faça a configuração normal dos componentes para exibir os dados no *DBGrid*. Para o envio automático da lista com os erros para a equipe de desenvolvimento, adicionamos no formulário os componentes *idSMTP* da paleta *IndyClients* e *idMessage* da paleta *Indy Misc* (o código será visto adiante no artigo).

### Cadastrando erros

Para reportar manualmente erros ou até mesmo solicitar melhorias no sistema, um novo formulário deverá ser construído, repetindo os mesmos passos da criação do formulário anterior, salvando a *unit* como "ufrmCadastroErro.pas".

Esse formulário foi criado seguindo algumas características da ferramenta *Trac* (Figura 8).

Ele também será usado para exibir os dados de um erro já cadastrado no banco. O evento *OnClick* do *Confirmar* deve ser implementado conforme a **Listagem 5**, que é bem parecido com a *CapturarErro* mas, ao contrário dessa função que registra automaticamente os problemas ocorridos, esse evento gravará os dados informados manualmente pelo cliente.

No *OnShow* vamos verificar se queremos cadastrar ou exibir um erro, utilizando o código da **Listagem 6**.

As variáveis *pCodErro* e *pOperacao* são do tipo *string* e devem ser declaradas na seção *private*. Por fim, implemente o método *Parâmetro* no formulário, conforme o código da **Listagem 7**.

Agora, com as janelas do sistema prontas, devemos programar no evento *OnClick* do *MenuItem* em *ufrmPrincipal* o código da **Listagem 8**, para exibir o Cadastro de Cliente e fazer o mesmo para exibir a janela de Lista de Erros, mas é necessário alterar o nome do formulário para "frmListaErros".

Fazer o mesmo na *ufrmListaErros* para exibir a tela de Cadastro de Erros, alterando o nome do formulário para "frmCadastroErro".

Para o formulário de listagem de erros, para chamar o mesmo, para adicionar um novo erro, devemos utilizar o código da **Listagem 9**.

Se quisermos visualizar as informações do erro, no item de menu *Consultar*, use o

código da **Listagem 10**.

Antes de executar a aplicação, vá na opção *Project>Options*, na aba *Form* do *Project Options for BugTracking.exe* deixe na lista *Auto-create forms* somente o *dtmdlPrincipal* e o *frmPrincipal*. Compile e execute o projeto.

No menu da janela principal, abra o Cadastro de Cliente e preencha todos os campos mas, no campo N° do Logradouro digite um valor não inteiro, e confirme os dados. No momento que clicar no *Confirmar*, o código implementado no evento *OnClick* tentará gravar um valor não inteiro no campo do tipo INTEGER no banco de dados, ocorrendo um erro não tratado pelo sistema (**Figura 9**). Assim o *CapturarErro* será chamado, gravando o erro automaticamente no banco de dados.

Para visualizar o erro, clique na opção *Reportar Erro* no menu da janela principal do sistema. Os erros existentes serão exibidos em uma lista, podendo também visualizar os detalhes de um erro, selecionando-o na lista e acessando o menu *Consultar*, da janela Lista de Erros (**Figura 10**).

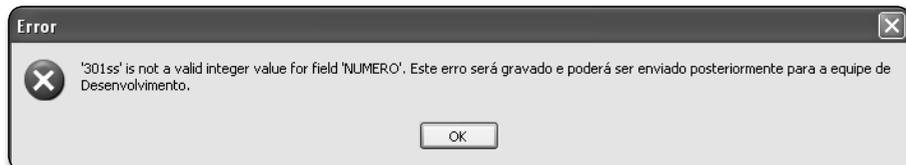
Se o usuário necessitar reportar uma nova necessidade para o sistema, como por exemplo, adicionar o campo CPF na tela de Cadastro de Cliente, também será possível através da tela de Cadastro de Erro, bastando selecionar no campo Tipo o valor *Nova Solicitação* e preencher os campos com as características de sua

**Listagem 6.** Exibindo ou cadastrando um erro

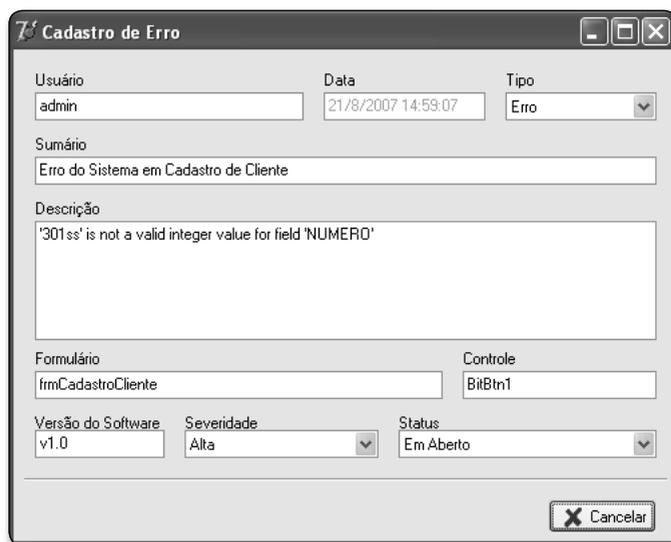
```
procedure TfrmCadastroErro.FormShow(Sender: TObject);
begin
  inherited;
  if pOperacao = 'Novo' then
  begin
    edtData.Text := DateTimeToStr(now);
    edtUsuario.Text := 'admin';
    edtVersao.Text := 'v1.0';
  end
  else
  begin
    pnlConsulta.Enabled := False;
    btnConfirmar.Visible := False;
    dtmdlPrincipal.clnIntdtstPrincipal.Close;
    dtmdlPrincipal.sqlIntdtstPrincipal.CommandText := 'select * from ERRO where ID_ERRO = '+ pCodErro;
    dtmdlPrincipal.clnIntdtstPrincipal.Open;
    edtUsuario.Text := dtmdlPrincipal.clnIntdtstPrincipal.FieldByName('USUARIO').AsString;
    edtData.Text := dtmdlPrincipal.clnIntdtstPrincipal.FieldByName('DATA').AsString;
    if dtmdlPrincipal.clnIntdtstPrincipal.FieldByName('TIPO').AsInteger <> -1 then
    begin
      cmbTipo.ItemIndex := dtmdlPrincipal.clnIntdtstPrincipal.FieldByName('TIPO').AsInteger;
    end;
    edtSumario.Text := dtmdlPrincipal.clnIntdtstPrincipal.FieldByName('SUMARIO').AsString;
    Descricao.Lines.Add(dtmdlPrincipal.clnIntdtstPrincipal.FieldByName('DESCRICAO').AsString);
    edtFormulario.Text := dtmdlPrincipal.clnIntdtstPrincipal.FieldByName('FORMULARIO').AsString;
    edtControle.Text := dtmdlPrincipal.clnIntdtstPrincipal.FieldByName('CONTROLE').AsString;
    edtVersao.Text := dtmdlPrincipal.clnIntdtstPrincipal.FieldByName('VERSAO').AsString;
    if dtmdlPrincipal.clnIntdtstPrincipal.FieldByName('STATUS').AsInteger <> -1 then
    begin
      cmbStatus.ItemIndex := dtmdlPrincipal.clnIntdtstPrincipal.FieldByName('STATUS').AsInteger;
    end;
    if dtmdlPrincipal.clnIntdtstPrincipal.FieldByName('SEVERIDADE').AsInteger <> -1 then
    begin
      cmbSeveridade.ItemIndex :=
        dtmdlPrincipal.clnIntdtstPrincipal.FieldByName('SEVERIDADE').AsInteger;
    end;
  end;
end;
end;
```

**Listagem 7.** Configurando o parâmetro para exibir ou cadastrar um erro

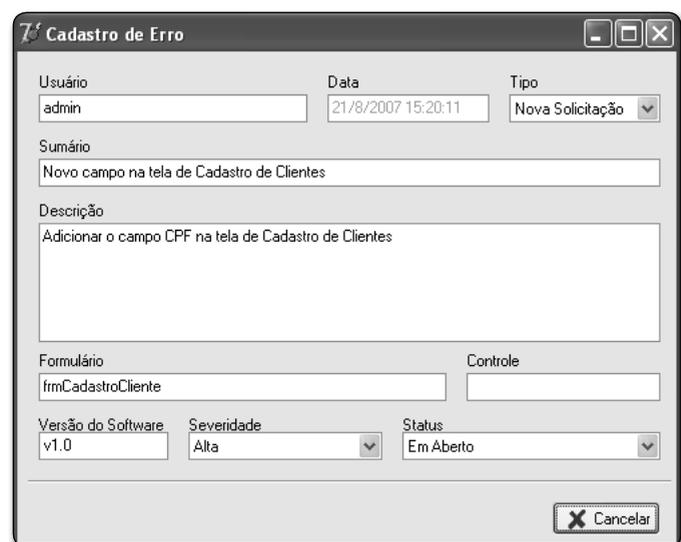
```
procedure TfrmCadastroErro.Parametro(CodErro, Operacao: string);
begin
  pCodErro := CodErro;
  pOperacao := Operacao;
end;
```



**Figura 9.** Visualização da mensagem do erro gerada



**Figura 10.** Visualização de um erro



**Figura 11.** Visualização dos erros gerados

### Listagem 8. Evento OnClick do MenuItem

```
uses ufrmCadastroCliente, ufrmListaErros;
...
procedure TfrmPrincipal.CadastrodeCliente1Click(Sender: TObject);
begin
  inherited;
  try
    frmCadastroCliente := TfrmCadastroCliente.Create(Self);
    frmCadastroCliente.ShowModal;
  finally
    frmCadastroCliente.Release;
  end;
end;
procedure TfrmPrincipal.ReportarErros1Click(Sender: TObject);
begin
  inherited;
  try
    frmListaErros := TfrmListaErros.Create(Self);
    frmListaErros.ShowModal;
  finally
    frmListaErros.Release;
  end;
end;
end;
```

### Listagem 9. Chamando o formulário para cadastro de erro

```
try
  frmCadastroErro := TfrmCadastroErro.Create(Self);
  frmCadastroErro.Parametro('0', 'Novo');
  frmCadastroErro.ShowModal;
finally
  frmCadastroErro.Release;
end;
```

### Listagem 10. Consultando um erro cadastrado

```
if cIntdtstGrade.FieldByName('ID_ERRO').IsNull then
begin
  Application.MessageBox(
    'É necessário selecionar um erro na lista.', 'Bug Report', MB_ICONWARNING);
end
else
begin
  try
    frmCadastroErro := TfrmCadastroErro.Create(Self);
    frmCadastroErro.Parametro(cIntdtstGrade.FieldByName('ID_ERRO').AsString, 'Consultar');
    frmCadastroErro.ShowModal;
  finally
    frmCadastroErro.Release;
  end;
end;
end;
```

### Listagem 11. Enviar e-mail com os dados dos erros

```
procedure TfrmListaErros.btnEnviarClick(Sender: TObject);
var
  bug:TextFile;
  i: Integer;
begin
  try
    cIntdtstGrade.First;
    AssignFile(bug, ExtractFilePath(Application.ExeName)+'\ListaErros.bug');
    Rewrite(bug);
    while not cIntdtstGrade.Eof do
      begin
        if cIntdtstGrade.FieldByName('STATUS').AsInteger = 0 then
          begin
            for i:= 0 to cIntdtstGrade.Fields.Count-1 do
              begin
                Write(bug, cIntdtstGrade.Fields[i].Value);
                Write(bug, ',');
              end;
            end;
            cIntdtstGrade.Next;
          end;
        CloseFile(bug);
        smtpEmail.Host := 'smtp...';
        smtpEmail.Port := 25;
        smtpEmail.Username := 'login de email do usuário';
        smtpEmail.Password := 'senha do usuário';
        smtpEmail.AuthenticationType := atLogin;
        msgEmail.From.Text := 'email do usuário';
        msgEmail.Recipients.EmailAddresses := 'email da equipe de desenvolvimento';
        msgEmail.Priority := mpHigh;
        msgEmail.Subject := 'Lista de Erros';
        msgEmail.Body.Text := 'Segue em anexo erros do sistema Bug Tracking.';
        TIdAttachment.create(msgEmail.MessageParts,
          TFileName(ExtractFilePath(Application.ExeName)+'\ListaErros.bug'));
        smtpEmail.Connect;
        try
          smtpEmail.Send(msgEmail);
        finally
          smtpEmail.Disconnect;
        end;
        Application.MessageBox('Email enviado com sucesso!', 'Bug Tracking',MB_ICONINFORMATION + MB_OK);
      except
        Application.MessageBox('Ocorreu um problema ao enviar o email.',
          'Bug Tracking',MB_ICONERROR + MB_OK);
      end;
    end;
  end;
```

nova necessidade.

Todos os erros, gerados automaticamente ou informados manualmente, bem como as novas solicitações cadastradas são exibidas na lista de erros, conforme **Figura 11**.

Para enviar tudo o que foi reportado pelo sistema para a equipe de desenvolvimento por e-mail, basta implementar no evento *OnClick* do *Enviar* do *frmListaErros* o código apresentado na **Listagem 11**.

O código gerará um arquivo com a extensão **BUG** com as informações dos erros e novas solicitações geradas no sistema, mas somente serão gravados no arquivo os registros que possuem Status *Em Aberto*, permitindo ao usuário o acompanhamento do que foi reportado.

O arquivo *ListaErros.bug* será gravado no mesmo local de onde o sistema está sendo executado, e enviará o mesmo automaticamente para a equipe de desenvolvimento do software por e-mail.

---

**Nota:** Configure as contas de e-mail no código do evento *OnClick* do *Confirmar*. Dependendo da conta de e-mail utilizada será necessário atualizar o Indy para uma versão mais atual.

---

## Conclusão

A comunicação é um fator que não pode ser desconsiderado ao longo de todo o ciclo de vida de um software. Mesmo depois da entrega do produto ao usuário final é desejável permitir interação com o projeto.

O uso de ferramentas para o rastreamento e o controle das mudanças ocorridas no projeto é um passo importante, gerando melhoria na qualidade tanto no processo de desenvolvimento do software quanto no produto gerado. ●