Boas práticas

Parte decisiva em manter um código limpo e de fácil manutenção, as boas práticas são essenciais para trabalhar em equipe em qualquer projeto. Neste capítulo, se encontram sugestões e práticas adotadas para a manutenção do código e uso das ferramentas utilizadas durante todo o processo de desenvolvimento.

Versionamento

Mensagens de Versionamento

Ao realizar qualquer alteração que seja necessário realizar um commit, é importantíssimo adicionar uma mensagem que possa informar do que se trata aquela alteração. Essa mensagem deve ser breve e descritiva, explicando em poucas palavras o que foi alterado, de forma que, caso outro desenvolvedor necessite realizar uma busca pelo log de alterações, as mensagens possam servir de "filtro" para identificar o que é relevante para a sua consulta.

> Deve-se evitar ao máximo realizar commits sem mensagem, principalmente em versões e no trunk.

Exemplos

Modelo de Mensagem

```
Caso XXXXX: Descrição do caso de acordo com o Mantis/Trello.

- Mensagem descritiva e em poucas palavras do que foi alterado

(Opcional) Desenvolvedor
```

Criação de branch

> Ao criar um branch, deve-se informar o número do caso e a sua descrição de acordo com o Mantis/Trello.

```
Caso XXXXX: Descrição do caso de acordo com o Mantis/Trello.

- Criação do branch

(Opcional) Desenvolvedor
```

Alterações

>

> Ao realizar o commit de alterações, além do cabeçalho indicando o número do caso e a descrição, deve-se adicionar por tópico as alterações feitas naquele commit.

```
Caso XXXXX: Descrição do caso de acordo com o Mantis/Trello.

- Alterações na lógica da rotina CalcularSaldo.
- Correções no padrão de código aplicado.
- Criação da classe TGerador

(Opcional) Desenvolvedor
```

Atualização de Branch

> Ao realizar a atualização de um branch(Merge com Trunk) , deve-se adicionar a "tag" [Atualização do Branch] no início do cabeçalho, para que ao consultar o log de alterações, o desenvolvedor identifique prontamente quais revisões são referentes à atualizações do branch e quais não são.

```
[Atualização do Branch] Caso XXXXX: Descrição do caso de acordo com o Mantis/Trello.

(Opcional) Desenvolvedor
```

Documentação Interna

Comentários

Segundo Robert C. Martin em Código Limpo: Habilidades Práticas do Agile Software,

> “O uso adequado de comentários é compensar nosso fracasso em nos expressar no código.”

Dessa maneira, o **uso de comentários no código deve ser evitado**, visto que, conforme o código passar por manutenção, a tendência é que o comentário fique desatualizado e não condizente com o trecho de código à que se refere.

Comentários que podem ser adicionados:

Comentários descritivos:

```
// INSS
if (DM_GERACAO.EventosSendoGerados.IndexOf('e301') <> -1) then
begin
Código Referente ao INSS
end;
```

> Comentários que complementam o entendimento de uma rotina, mas que não tentam explicar o que está sendo feito no trecho.

Comentários que devem ser evitados/removidos:

Comentários no cabeçalho:

```
{**************************************************************
* Módulo : ArqTeste ***********
```

```
* Finalidade : Realizar a exportação de Notas Fiscais □□□*
* de Serviços para Prefeitura de São Paulo □□□□□□*
* Data : 21/02/2000 □□□□□□□□□□*
* Programador : Fulano da Silva Santos □□□□□□□*
**********************************************************}
```

> Comentários utilizados como cabeçalho que descrevem o que a unit faz, a data de implementação e o programador responsável pela implementação apesar de serem descritivos, muitas vezes não são atualizados conforme o código passa por mudanças. Essas informações podem e devem ser encontradas nas mensagens de versionamento.

Comentários referentes à processos passados:

```
// MIGRACAO
// Caso a classe uClass.NumDocs seja utilizada em alguma unit
// a diretiva referente ao arquivo uClass.NumDocs.inc deverá ser acrescentada.
// Ex: Ver units FormPri e Backup, antes do nome da unit;
// No arquivo uClass.NumDocs.inc deverá ser adicionada a diretiva referente ao
// sistema ao qual a classe NumDocs foi adicionada.
```

> Comentários como esses não acrescentam informações no entendimento do código e acabam criando uma poluição visual na unit, de modo que, com o passar do tempo a tendência é que esse tipo de comentário seja automaticamente ignorado pelos desenvolvedores.

Código comentado:

```
// if ((QMANUT.FieldByName('DiasDireito').AsFloat = 0) or
//□   (SameText(Trim(QMANUT.FieldByName('DiasDireito').AsString), ''))) and
//     (MessageDialog.Show('Não foi informado os dias de Direito. Deseja Continuar?',
//□□  mtConfirmation, [mbYes, mbNo], 0) = mrNo) then
// begin
// □ QMANUT.FieldByName('DiasDireito').FocusControl;
// □ Abort;
// end;
```

> Códigos comentados geram confusões no código, além de poluir desnecessariamente a unit, dessa maneira códigos não devem ser comentados e sim removidos. Caso necessário, pode-

se visitar o histórico de revisões para acompanhar as alterações do trecho.

Código Limpo

Definição de Clean Code

A apresentação de um código claro e organizado não consiste apenas na convenção nomes, constantes, classes, variáveis, espaçamento etc.. Um código limpo (*clean code*) deve ser:

- Simples: fácil entendimento;
- Eficiente: realizar tudo o que foi proposto;
- Único: não realizar algo que outro trecho de código já faz;
- Direto: não dar voltas para chegar no resultado;
- Feito com atenção: o código deve ser sempre feito com preocupação e revisto depois de pronto;

Bad Smells

Em contra partida, também existem as *Bad Smells*, que como sua tradução já sugere, é algo com cheiro ruim, e representa o código com práticas que não devem ser utilizadas. Essas práticas se devem aos códigos que fogem das características de um código limpo. Abaixo são demonstrados alguns exemplos de *bad smells*.

Encadeamento de If's

```
if (Q_Manut.FieldByName('DiasDireito').AsFloat = 0) then
begin
  if (AnsiSameText(Trim(Q_Manut.FieldByName('DiasDireito').AsString), '')) then
  begin
    if (MessageDialog.Show('Não foi informado os dias de Direito. Deseja' +
          'Continuar?', mtConfirmation, [mbYes, mbNo], 0) = mrNo) then
    begin
      Q_Manut.FieldByName('DiasDireito').FocusControl;
      Abort;
    end;
  end;
end;
```

> Esse tipo de encadeamento pode ser facilmente substituído pela palavra reservada "and".

```
if (Q_Manut.FieldByName('DiasDireito').AsFloat = 0) and
   (AnsiSameText(Trim(Q_Manut.FieldByName('DiasDireito').AsString), ''))) and
   (MessageDialog.Show('Não foi informado os dias de Direito. Deseja' +
      'Continuar?', mtConfirmation, [mbYes, mbNo], 0) = mrNo) then
begin
  Q_Manut.FieldByName('DiasDireito').FocusControl;
  Abort;
end;
```

Excesso de if-else

```
if (Q_Manut.FieldByName('DiasDireito').AsFloat = 0) then
  LTipo := 1
else if (Q_Manut.FieldByName('DiasDireito').AsFloat = 1) then
  LTipo := 2
else if (Q_Manut.FieldByName('DiasDireito').AsFloat = 2) then
  LTipo := 3
else if (Q_Manut.FieldByName('DiasDireito').AsFloat = 3) then
  LTipo := 5
else
  LTipo := 9;
```

> O trecho acima pode ter sua estrutura facilitada com a utilização de um "case".

```
case Q_Manut.FieldByName('DiasDireito').AsFloat of
  0: LTipo := 1;
  1: LTipo := 2;
  2: LTipo := 3;
  3: LTipo := 5;
  else
    LTipo := 9;
end;
```

Atribuição Indireta

> Situações com atribuições indiretas aparecem constantemente no código

```
if (Q_Manut.FieldByName('DiasDireito').AsFloat = 0) then
  LDeveIncrementar := False
else
  LDeveIncrementar := True;
```

```
if (RG_TipoCliente.ItemIndex = 0) then
  LTipo := 0
else (RG_TipoCliente.ItemIndex = 1) then
  LTipo := 1;
```

> Basta uma pequena análise por parte do desenvolvedor para perceber que podem ser simplificadas

```
 LDeveIncrementar := (Q_Manut.FieldByName('DiasDireito').AsFloat <> 0);
```

```
LTipo := RG_TipoCliente.ItemIndex;
```