INSTITUTO FEDERAL DE
EDUCAÇÃO CIÊNCIA E TECNOLOGIA
Baiano

DOCUMENTO DE PADRONIZAÇÃO

METODOLOGIA DE DESENVOLVIMENTO DE SISTEMAS

– FASE DA ESPECIFICAÇÃO –

COORDENAÇÃO DE SISTEMAS (CODES)

JUNHO/2011

Rua do Rouxinol, N° 115 / Salvador – Bahia –CEP: 41.720-052
Telefone: (71) 3186-0001. Email: codes@ifbaiano.edu.br
Site: http://www.ifbaiano.edu.br

INSTITUTO FEDERAL DE
EDUCAÇÃO CIÊNCIA E TECNOLOGIA
Baiano

SUMÁRIO

1. Controle de Versão

Data	Versão	Descrição	Autor
[dd/mm/aaaa]	[1.0, 1.1, ...]	[Inserir ação no documento que gerou a nova versão]	[Nome do Responsável pela Atualização da Versão]

2. Objetivo

Esse documento visa padronizar a forma de desenvolvimento realizada no IF Baiano a fim de:

- Reduzir os riscos no desenvolvimento;
- Aumentar a confiabilidade e fácil manutenção do *software*;
- Auxiliar o programador;
- Aumentar a produtividade e agilidade da equipe.

Mesmo tentando estabelecer padrões rígidos e detalhados, em diversas situações não será possível aplicar integralmente o padrão geral, seja por limitações de espaço em tela, por particularidades do programa ou pela facilidade de operação que um desvio de padrão poderá conferir ao programa. Em todo caso, porém, um desvio no padrão definido nesse documento deverá ser discutido previamente com o analista responsável.

3. Padrões de Codificação

3.1. Nome de arquivos

Classes: Nome da classe com a primeira letra de cada palavra em maiúsculo;

Outros arquivos: 02 letras para o tipo do artefato + mnemônico identificador com a primeira letra em maiúsculo.

3.2. Organização de arquivos

Um arquivo consiste de seções que devem ser separadas por linhas em branco e um comentário opcional identificando cada seção.
Evitar a criação de arquivos maiores que 2000 linhas.

3.3. Localização dos arquivos

Todos os arquivos devem ser armazenados em repositório único do projeto através de um sistema de controle de versões.

3.4. Estrutura dos arquivos

Todos os arquivos devem conter:

- Cabeçalho (Comentários de Documentação);
- Declarações;
- Instruções comentadas (Comentários de Implementação).

3.5. Endentação

A endentação é um dos principais elementos para tornar um código claro e de fácil leitura. A endentação deve ser composta por 04 espaços.

3.6. Comprimento das linhas

Evitar comprimentos de linhas maiores que 80 caracteres já que muitas ferramentas e terminais não trabalham bem com linhas maiores que isso.

3.7. Comentários

Programas em geral podem ter dois tipos de comentários: comentários de implementação e comentários de documentação.

3.7.1. Comentários de implementação

Comentários de implementação ou simplesmente comentários são aqueles específicos para descrever especificamente a implementação de alguma coisa no código.

Os comentários são feitos para dar uma visão geral do código e prover informações adicionais que não estão prontamente disponíveis no próprio código. Apenas devem ter informações que ajudem a ler e a entender o funcionamento do programa.

Comentários não devem ser colocados em grandes caixas desenhadas com asteriscos ou outros caracteres.

3.7.2. Comentários de documentação

Comentários de documentação são aqueles que visam documentar a especificação do código, em uma visão livre da implementação, pois são feitas para serem lidas por programadores que podem não ter o código fonte na mão. Em algumas linguagens permite gerar documentações automatizadas do programa com o uso de programas específicos.
Não deve haver mais de um comentário por classe, interface ou membro. Os comentários devem aparecer logo antes da declaração.

3.8. Declarações

Esta seção é destinada a descrição de como fazer declarações de variáveis, classes, interfaces e métodos/funções/procedimentos (nomes e regras gerais).

3.8.1. Número por linha

É recomendada apenas uma declaração por linha, porque encoraja o comentário, por consequência não declare diferentes tipos na mesma linha,

3.8.2. Inicialização

Tente iniciar variáveis locais onde elas forem declaradas. A única razão para não iniciar uma variável onde ela foi declarada, é se o seu valor inicial depende que alguma computação seja feita antes.

3.8.3. Posicionamento

Coloque declarações somente no início de cada bloco de código. Não espere para declarar variáveis até seu primeiro uso; isso pode confundir os programadores e dificultar a portabilidade dentro de um escopo.
Evite utilizar declarações locais que escondam declarações de um nível maior, ou seja, não declare variáveis de mesmo nome em um bloco interno:

3.8.4. Declaração de Blocos

Quando programando, as seguintes regras devem ser seguidas:

- Não colocar espaço entre o nome do método/função/procedimento e o parênteses "(" iniciando sua lista de parâmetros.

- A abertura da chave deve ser no fim da mesma linha que a declaração da instrução.
- O fechamento da chave deve ser na linha subsequente a última linha do bloco, sozinha e endentada, alinhando com a instrução do início do bloco, exceto quando uma instrução vazia, onde o fechamento da chave deve aparecer imediatamente depois da sua abertura.

3.9. Instruções

Esta seção descreve como proceder na escrita das instruções condicionais e de controle.

3.9.1. Instruções simples

Cada linha deve conter no máximo uma instrução.

3.9.2. Instruções compostas

Instruções compostas são instruções que contem várias instruções aninhadas.

- A instrução aninhada deve ser endentada um nível acima do que a instrução composta.
- A chave de abertura deve ficar no final da linha que começa a instrução composta; a chave de fechamento deve começar uma linha e ter o mesmo nível de endentação da instrução composta.
- Chaves devem ser usadas blocando todas as instruções. Isso torna mais fácil adicionar instruções sem acidentalmente introduzir novos *bugs* pelo fato de se esquecer de colocar as chaves.

3.10. Espaços em branco

Aqui são descritos os procedimentos para utilização de espaços em branco ao longo do código fonte.

3.10.1. Linhas em branco

Linhas em branco melhoram a legibilidade separando seções do código que são logicamente relacionadas.

Duas linhas em branco devem sempre ser usadas nas seguintes circunstâncias:

- Entre seções em um arquivo fonte;

Uma linha em branco deve sempre ser usada nas seguintes circunstâncias:

- Entre métodos/funções/procedimentos;
- Entre as variáveis locais em um método e sua primeira declaração;
- Antes de um comentário em bloco ou um comentário de uma linha;
- Entre seções lógicas dentro de um método para melhorar a legibilidade.

3.10.2. Espaços em branco

Espaços em brancos devem ser usados nas seguintes circunstâncias:

- Uma palavra-chave seguida de parênteses deve ser separada por um espaço.
- Note que um espaço em branco não deve ser usado entre um nome de métodos/funções/procedimentos e seus parênteses. Isso ajuda na diferenciação entre métodos/funções/procedimentos e palavras-chave.
- Um espaço em branco deve aparecer após vírgulas nas listas de argumentos.
- Todos os operadores binários, exceto (ponto), devem ser separados de seus operandos por espaços. Espaços em branco nunca devem separar operadores unários, como incremento (“++”) e decremento (“--“), de seus operandos.
- Um espaço em branco deve aparecer após ponto e vírgula nas expressões.
- *Casts* devem ser seguidos por um espaço em branco.

3.11. Convenção de nomes

Padrões de nomenclatura tornam os programas mais compreensíveis fazendo-os mais fáceis de ler. Eles podem também dar informações sobre a função do identificador, por exemplo, se é uma constante, variável ou método/função/procedimento, o que pode ser útil no entendimento do código.

3.11.1. Métodos

Métodos devem ser verbos com a primeira letra minúscula. Em casos compostos a primeira letra de cada palavra interna deve ser maiúscula.

3.11.2. Variáveis

Nomes de variáveis não devem começar com sublinhado (“_”) ou cifrão (“$”), embora os dois sejam permitidos.

Nomes de variáveis devem ser curtos, mas ainda assim significativos. Em casos compostos a primeira letra de cada palavra interna deve ser maiúscula. A escolha do nome da variável deve ser projetada para indicar ao observador casual a intenção de seu uso.

Nomes de variáveis de uma letra somente devem ser evitados exceto para variáveis temporárias como contadores e *flags*. Nomes comuns para variáveis temporárias são “i”, “j”, “k”, “m”, e “n” para inteiros; “c”, “d”, e “e” para caracteres.

3.11.3. Constantes

Os nomes das variáveis declaradas como constantes devem ser todo escrito com maiúsculas e com as palavras separadas por sublinhado (“_”).

3.12. Práticas de programação

Aqui são apresentadas “boas práticas de programação”. Boas práticas é uma expressão derivada do inglês "*best practices*" que denomina técnicas identificadas como as melhores para realizar determinada tarefa, neste caso, a tarefa de programação.

3.12.1. Escopo de Variáveis e Métodos/Funções/Procedimentos

Não torne qualquer variável, método/função/procedimento público sem um bom motivo.

3.12.2. Constantes

Constantes numéricas (literais) não devem ser codificadas diretamente, exceto para -1, 0 e 1, que podem aparecer em um *loop for* como valores de controle.

3.12.3. Atribuições de Variáveis

Evite atribuir várias variáveis com o mesmo valor em uma única declaração. Isso é difícil de ler.

Não use o operador de atribuição em um lugar onde ele pode ser facilmente confundido com o operador de igualdade.

3.12.4. Práticas Diversas

3.12.4.1. Parênteses

É geralmente uma boa idéia usar parênteses liberalmente em expressões envolvendo operadores mistos para evitar problemas de precedência de operadores. Mesmo que a precedência de operadores pareça clara para você, ela pode não ser para outros – você não deve presumir que ouros programadores saibam procedências, assim como você.

4. Banco de dados

Esta seção estabelece políticas de uso dos ambientes de banco de dados envolvidos no IF Baiano, definindo regras para nomenclatura dos objetos e dados armazenados para garantir seu compartilhamento e consistência.

4.1. Criação e Alteração de Objetos de Banco de Dados

Toda criação ou alteração de Objetos de banco de dados deve ser armazenada em *scripts*. Esses *scripts* estão organizados conforme a sua utilização. A utilização está classificada da seguinte forma:

- Visões;
- Restrições (constraints);
- Permissões aos objetos;
- Índices para as tabelas;
- Sequências;
- Sinônimos dos objetos;
- Tabelas;
- Triggers.

4.2. Tabelas

Todos os nomes de tabelas devem começar com a primeira letra maiúscula. Em casos compostos cada palavra interna deve ter a primeira letra maiúscula. A primeira parte do nome deve ser referente ao contexto ao qual a tabela pertence dentro do programa (03 caracteres). A segunda parte deve ser o nome propriamente dito da tabela, ou seja, seu nome funcional. A divisão das partes que compõem o nome da tabela deve ser feita por um sublinhado (“_”).

4.3. Colunas ou Campos

Todos os nomes de colunas devem ser totalmente em letras minúsculas. Devem-se evitar nomes compostos ou muito extensos quando não for estritamente necessário, caso haja a necessidade todas as palavras internas devem ter a primeira letra maiúscula.
Obs.: Toda tabela deve ter uma coluna contendo um identificador único do tipo inteiro.

4.4. Chaves Estrangeiras

As chaves estrangeiras devem ter exatamente o mesmo tipo da chave primária a qual elas se referenciam. Além disso, um sufixo com o nome da tabela a qual a chave se referencia deve ser anexado ao nome por meio de um sublinhado (“_”) e deve ser escrito em letras minúsculas.

4.5. Campos Padrão

Todas as entidades deverão possuir os seguintes campos padrão para o controle de atualização dos dados.

CAMPO	TIPO	DESCRIÇÃO
data_criacao	DATETIME	Data de criação
usuario_criacao	VARCHAR(100)	Usuário da criação
data_ultima_atualizacao	DATETIME	Data última atualização
usuario_ultima_atualizacao	DATETIME	Usuário última atualização

É obrigatório que a manutenção das informações padrões das entidades seja prevista nos programas desenvolvidos.

4.6. Sequências

O nome deve ser em português e formado pelo nome da tabela, finalizado pelo sufixo “_SEQ”.

4.7. Visão

O nome deve ser em português, seguindo as mesmas regras da criação de tabelas. Porém ao final do nome deve ser incluido o sufixo “_V”.

4.8. Procedimentos

O nome deve ser em português e formado por um nome significativo que identifique a função do procedimento, seguido pelo sufixo “_PRC”.

4.9. Funções

O nome deve ser em português e ser um nome significativo que identifique o objetivo da função, seguido pelo sufixo “_FN”.

4.10. Declarações SQL

Todas as tabelas e colunas devem ser criadas com cláusulas que respeitem os padrões SQL ANSI. Deve-se evitar usar estruturas, como por exemplo, tipos de dados, proprietárias ou específicas de certo SGBD. Com isso a portabilidade fica garantida.

Todas as cláusulas SQL devem ser escritas totalmente em letras maiúsculas como nos exemplos anteriores. Assim é mais fácil diferenciar as cláusulas dos dados.

5. Versionamento

Um sistema de controle de versão é responsável por permitir que pessoas trabalhem em um mesmo projeto, editando seus arquivos sem que uma pessoa sobrescreva as alterações das outras.

5.1. Por que um Sistema de Controle de Versão é Usado

Porque guarda todos os arquivos do projeto, e todas as mudanças que já foram feitas a esses arquivos. Isso permite que as mudanças sejam revisadas ou desfeitas e que o arquivo do projeto seja resgatado após qualquer mudança dentre outras possibilidades.

O Sistema de Controle de Versão permite que vários desenvolvedores trabalhem juntos no mesmo projeto, e até mesmo trabalhem no mesmo arquivo ao mesmo tempo. Ele registra:

- O que foi mudado;
- Quando foi mudado;
- Quem mudou;
- Porque foi mudado.

5.2. Estrutura do projeto

O projeto é estruturado em *trunk* e *branch*. O *trunk* é a versão do sistema mais atualizada, é onde os desenvolvedores devem enviar seus arquivos. Já o *branch* é onde ficam os candidatos a *release* de versões, sempre que o sistema for gerar uma versão nova deve ser feito um *branching* a partir do *trunk* com o número da versão a ser lançada, por exemplo, *branches*/1.0.

Após a versão candidata ser testada e aprovada para publicação, ela deve gerar uma *tag* com o número da versão.

Outro uso do *branch* é quando um desenvolvedor pretende fazer alguma alteração grande o bastante que pode desestabilizar o sistema enquanto a mudança é feita, dessa forma o desenvolvedor cria um *branch* enquanto faz as mudanças e depois quando terminar integra as mudanças ao *trunk*.

5.3. Recomendações aos desenvolvedores

A seguir, seguem algumas recomendações importantes para a utilização de um Sistema de Controle de Versão.

5.3.1. Atualize com freqüência

Evite conflitos estando sempre atualizado com o repositório. Quanto mais tempo você demorar em atualizá-lo, mais difícil irá ser para o *software* mesclar suas modificações com as modificações feitas por outros desenvolvedores.

5.3.2. Comunique-se

Não comece a trabalhar em algo grande sem avisar ninguém. Nada é mais irritante do que descobrir que você e outra pessoa estavam trabalhando o dia todo no mesmo problema.

5.3.3. Pense duas vezes

Qualquer alteração no Sistema de Controle de Versão pode ser desfeita, mas se você enviar um código quebrado, esse código poderá se espalhar e causar problemas.

Atualize antes de enviar seus arquivos para ter certeza que eles estão atualizados. Do contrário, você está arriscando sobrescrever mudanças que outras pessoas fizeram nesse meio tempo.

Se tiver dúvida sobre suas alterações, cheque as diferenças do seu código com o repositório.

5.3.4. Envie seus arquivos regularmente

Envie seus arquivos toda vez que estiver com medo de perder seu trabalho. Sempre envie os arquivos quando terminar modificações.

Não misture problemas não relacionados no mesmo envio.

Envios separados tornam as revisões mais fáceis, e se necessário retroceder mudanças específicas.

Se um único conserto alterar vários arquivos, então envie todos eles juntos.

Todo envio deve ter um comentário explicando o porquê das alterações.

Nunca bloqueie um arquivo, o sistema faz um bom trabalho na integração de mudanças, e isso permite um desenvolvimento paralelo muito melhor.

6. Referências

PROJETO DE ACESSIBILIDADE VIRTUAL. **Padrões e Procedimentos para Desenvolvimento.** Dezembro de 2009.