Universidade Federal de Goiás
Instituto de Informática
Curso: Engenharia de Software

Disciplina: Técnicas Avançadas em Construção de Software
2º semestre/2015

Modelo de documento de qualidade de código à ser implementado no desenvolvimento do software Sorteio de bilhetes

Prof. Msc. Marcelo Ricardo Quinta

Grupo:
Dyego de Oliveira
Osmar Cavalcante
Rafael Carvalho
Lucas Machado
Talles Marinho

Goiânia 2015

Boas práticas de Desenvolvimento

➢ Aspectos do Projeto

Projeto desenvolvido em Java (seguindo os bons costumes)

- ## Definição da Convenção de Nomes

Lista de Verificação: Atribuição de nomes de variáveis considerações gerais

1) **O nome descreve total e precisamente o que a variável representa?**

No projeto SORTEIO DE BILHETES estamos seguindo os bons costumes na linguagem Java com descrição clara sobre o que a variável representa, tendo em base a tabela 11-5 do livro de **MCCONNELL [1]**. Exemplos de convenções de atribuição de nomes Java.

2) **O nome se refere ao problema do mundo real, em vez de à solução da linguagem de programação?**

Sim, os nomes estão sendo atribuídos de forma que qualquer pessoa com o conhecimento mínimo da linguagem, possa entender sobre o que se refere certo trecho de código e também ter uma ideia sobre as funcionalidades de cada rotina.

3) **O nome é longo o suficiente para que você não tenha que decifrá-lo?**

Sim, de acordo com as tabelas 11-1 e 11-2 do livro **[1]**, os nomes das variáveis estão sendo atribuídas de forma que fiquem com o tamanho adequado e que não tenham significados obscuros.

4) **Os quantificadores de valor calculado, se houver, estão no final do nome?**

Sim, os quantificadores de valor estão sendo atribuídos ao final do nome da variável, como por exemplo: *sai_ValorMaximo, sai_ValorMinimo, sai_ValorMedio*

5) **O nome usa *Count* ou *Index*, em vez de *Num* ?**

Sim, pois estes nomes trazem uma melhor compreensão pelo fato de que estão padronizados internacionalmente como contadores, assim uma pessoa de outro país que for ler o código vai perceber que aquela variável está fazendo o papel de um contador dentro do contexto que ela está inserida.

➢ Atribuição de Nomes de Tipos de Dados Específicos

1) **Todas as variáveis "Temporárias" mudaram de nome para algo mais significativo?**

Sim, com o objetivo de esclarecer a verdadeira função da variável.

2) **As variáveis booleanas receberam nomes, de modo que seus significados sejam claros quando elas são verdadeiras?**

Sim, os nomes das variáveis booleanas estão atribuídas para que não ocorra confusões quanto ao seu significado, ex.: `Cancelar.setText("Encerrar")`.

3) **Os nomes de tipo enumerado incluem um prefixo ou sufixo que indicam a categoria - por exemplo: Color_para, Color_Red, Color_Green,Color_Blue, etc.?**

Sim, usaremos o padrão para linguagens OO que é *"ValorNumeroFaixas.setForeground(Color.BLUE)"*

4) **As constantes têm nomes de acordo com as entidades abstratas que representam, em vez dos números a que se referem?**

Sim, e também as constantes estão de acordo com o padrão, ex.: `numDeFaixas = 100.`

➢ Convenções de Atribuição de nomes

1) A convenção faz distinção entre dados locais, de classe e globais ?

Sim. Segundo os bons costumes da linguagem Java, as variáveis locais, quando referenciadas diretamente dentro da própria classe, utilizamos a referência 'this', como forma de auto-referenciar o próprio contexto em que se encontra.

2) A convenção faz distinção entre nomes de tipos, constantes nomeadas, tipos enumerados e variáveis?

Não necessariamente. Fica a cargo do responsável pela implementação criar variáveis com nomes legíveis, pronunciáveis e óbvios, que denotem o 'o que' a variável deveria representar. Ex: `VerificarErrosCampos();`

3) A convenção identifica parâmetros somente de entrada para rotinas nas linguagens que não os impõem ?

i. Não se aplica, ainda não foram encontrados exemplos do descrito.

4) **A convenção é o mais compatível possível com as convenções - padrão da linguagem ?**
    i. Sim. A convenção foi criada tendo como base os bons costumes da linguagem Java.

5) **Os nomes são formados de forma a dar legibilidade ?**
    i. Sim. Ex: *"nomeArquivoSaida"*

➢ Nomes Curtos

1) **O código usa nomes longos(a não ser que seja necessário usar nomes curtos) ?**
Depende, o código usa nomes semânticos isso depende muito, temos foco em entregar um código limpo e semântico.

2) **O código evita abreviações que economizam apenas um caractere?**
Sim. Abreviações aceitáveis com palavras resultantes que sejam autoexplicativas e no mínimo 3 caracteres. Ex: "*EfetuarSorteio.msgDeErro.length()*" , "*numBilhetes*", "*numPremios*".

3) **Todas as palavras estão abreviadas consistentemente?**
Sim. As abreviações são consistentes ao ponto de se entender a palavra inteira. Ex: *num (número)*, *msg (mensagem)*; sempre seguido de nomes que finalizam a caracterização da palavra.

4) **Os nomes podem ser pronunciados?**
Sim. Abreviações não pronunciáveis não podem ser compreendidas com olhar rápido, por isso não foram utilizadas.

5) **Os nomes que poderiam ser lidos ou pronunciados de forma errada foram evitados?**
Não se aplica. Não foram encontrados exemplos do descrito.

6) **Os nomes curtos estão documentados em tabelas de transporte (glossários)?**
Não. De acordo com a convenção, os nomes devem ser utilizados de forma que transmitam seu entendimento, ou seja, mesmo que os nomes sejam curtos

(abreviações), só será aceito caso tenha significado óbvio, fazendo com que a utilização de glossários seja desnecessária.

➢ Problemas de Atribuição de Nomes Comuns

1) Nomes enganosos?

Não. Na teoria, se todos os envolvidos seguirem as especificações deste atual documento, não existiram nomes enganosos ou ambíguos.

2) Nomes com significados semelhantes?

Não. Assim como exposto na questão anterior, visando manter o projeto em conformidade com o que recomendam os bons costumes, usamos abreviações de palavras para evitar grafias semelhantes.

3) Nomes que são diferentes apenas por um ou dois caracteres?

Não. Conforme determina os bons costumes em Java a possibilidade do uso de variáveis com até *20 caracteres* e abreviações de palavras previnem a ocorrência de nomes com grafia semelhante. Estes recursos são usados (não na sua totalidade) no nosso projeto.

4) Nomes que têm sons semelhantes?

Eventualmente Sim, mas com grafia diferente que é o fator determinante para facilitar a manutenção e documentação do código.

5) Nomes grafados de forma errada intencionalmente, para torná-los mais curtos?

Não, os recursos de abreviação e truncamento de palavras são usados no projeto para evitar tal problema.

6) Nomes que são normalmente grafados de forma errada em inglês (ou português)?

Não, os recursos usados na questão anterior associados a outros comuns em convenção de nomes previnem a ocorrência deste tipo de problema no projeto.

7) Nomes que entram em conflito com nomes de rotina da biblioteca - padrão ou com os nomes de variável predefinidos?

Não se aplica a este contexto, já que a convenção de nomes estipula apenas nomes legíveis, objetivos e descritivos, não se atentando exatamente ao contexto de todo o sistema, mas sim em partes específicas separadamente.

8) Nomes totalmente arbitrários?

A convenção de nomes desestimula a criação de nomes arbitrários, dando incentivo a criação de nomes objetivos e descritivos.

9) Caracteres difíceis de ler?

A convenção de nomes desestimula a criação de nomes de difícil leitura, dando incentivo a criação de nomes legíveis.

Tabela 1 - Classe EfetuarSorteio [2]

Tipo	Nome Antigo	Nome Novo
Variável	sai_ValorNumeroFaixas	numDeFaixas
Variável	sai_ValorEsperado	valorEsperado
Variável	sai_ValorDesvioPadrao	valorDesvioPadrao
Variável	sai_ValorMedio	valorMedio
Variável	sai_ValorMaximo	valorMaximo
Variável	sai_ValorMinimo	valorMinimo
Variável	sai_ValorDiferencaMaxMin	diferencaValoresMaxMin
Variável	sai_HashResultado	hashResultado
Variável	Vezes	vezes
Variável	Premios	premios
Variável	Nome	nomeSorteio
Variável	MsgErro	msgDeErro
Variável	PararSorteio	sorteioParado
Variável	EmExecucao	emExecucao
Variável	en_Extracao	extracao
Variável	en_DataExtracao	dataExtracao
Variável	en_Premio1	premio1
Variável	en_Premio2	premio2
Variável	en_Premio3	premio3
Variável	en_Premio4	premio4
Variável	en_Premio5	premio5
Variável	en_SorteioNumero	numDoSorteio
Variável	en_DataSorteio	dataSorteio
Variável	en_DataDiarioOficial	dataDiarioOficial
Variável	en_TotalBilhetes	totalDeBilhetes

Variável	en_totalPremios	totalDePremios
Variável	en_NomeArquivoTXT	nomeArquivoTxt
Variável	Resultado	resultado
Variável	NumBilhetes	numDeBilhetes
Variável	NumPremios	numDePremios
Variável	StrPremios	strPremios
Variável	StrBilhetes	strBilhetes
Variável	ValorMedioEsperado	valorMedioEsperado
Variável	valores	valorDosBilhetes
Variável	MetodoHASH	metodoHash
Variável	MEGABYTE	megabyte
Variável	Caminho	caminho
Variável	MemoriaNecessaria	memoriaNecessaria
Variável	TamMemoria	tamanhoMemoria
Variável	outS	outSWriter
Variável	out	printWriter
Variável	rg	rngDoSorteio
Variável	faixas	numFaixas
Variável	ri	chaveBilhetes
Variável	Max	maxPremio
Variável	Min	minPremio
Variável	SomaDosQuadrados	somaDosQuadrados
Variável	DesvioPadrao	desvioPadrao
Variável	DifMaxMin	difMaxMin
Função	abrir	abrirArquivoTxt
Função	Parar	pararSorteio

Tabela 2 - Classe Sorteio [2]

Tipo	Nome Antigo	Nome Novo
Variável	afr_Premio1	afrPremio1
Variável	afr_Premio2	afrPremio2
Variável	afr_Premio3	afrPremio3
Variável	afr_Premio4	afrPremio4
Variável	afr_Premio5	afrPremio5

Variável	afr_DataExtracao	afrDataExtracao
Variável	afr_Extracao	afrExtracao
Variável	afr_Sorteio	afrSorteio
Variável	afr_DataSorteio	afrDataSorteio
Variável	afr_DataDiarioOficial	afrDataDiarioOficial
Variável	afr_Bilhetes	afrBilhetes
Variável	afr_Premios	afrPremios
Variável	TimerSorteio	timerSorteio
Variável	FormatoNumero	formatoNumero
Variável	TempoInicial	tempoInicial
Variável	DeltaAnterior	deltaAnterior
Variável	NomeArquivoSAIDA	nomeArquivoSaida
Variável	ArquivosTXT	arquivosTxt
Variável	ProximaExtracao	proximaExtracao
Variável	NumeroFaixas	numDeFaixas
Variável	AferindoPrograma	programaAferido
Variável	Versao	numDoSorteio
Variável	Agora	dataLocal
Variável	dt	sdf
Variável	DeuErro	erroEncontrado
Variável	NumBilhetes	numBilhetes
Variável	NumPremios	numPremios
Variável	StrBilhetes	strBilhetes
Variável	TempoAgora	horaLocal
Variável	TempoFinal	tempoFinal
Variável	ContinuaExecucao	execucaoContinua
Função	InicializarResultados	inicializarResultados
Função	InicializarCampos	inicializarCampos
Função	CarregarValoresPadrao	carregarValoresPadrao
Função	HabilitarCampos	habilitarCampos
Função	TimerSorteio_Timer	iniciarTimer
Função	VerificarErrosCampos	verificarErrosCampos
Função	EfetuarUmSorteio	efetuarUmSorteio

➢ Métricas de qualidade de Código

1. ***CYCLO Cyclomatic Complexity:*** *[7]* Complexidade Ciclomática calcula o número de caminhos linearmente independentes no método analisado, para esse projeto o máximo é 7.

2. ***LOC (Lines of Code):*** *[5]* Calcula o número de linhas efetivas de um método. Linhas compostas apenas por comentários ou vazias não são consideradas efetivas para esse projeto o máximo é 50.

3. ***NOM (Number of Methods):*** A métrica NOM calcula o número de métodos de uma classe. Seu valor mínimo é zero e não existe um limite máximo para o seu resultado. Obtemos zero quando a classe não possui métodos no máximo 10.

4. ***SLOC:*** **(Soma do Número de linhas A métrica)** SLOC calcula a soma do número de linhas efetivas de todos os métodos de uma classe. Ou seja, seu resultado é a soma dos valores do LOC de cada método da classe no máximo 10.

5. **LCOM4 (Lack of Cohesion of Methods):** *[3]* Calcula a falta de coesão dos métodos da classe analisada e seu máximo e 2.

6. **NC (Number of Calls)** Números de chamada calcula o número de métodos e atributos *internos e externos* usados por um método.

7. **NEC (Number of External Calls):** *[6]* Número de Chamadas Externas calcula o número de métodos e atributos externos acessados por um método. NEC é uma adaptação da métrica *ATFD (Access To Foreing Data),* que calcula o número de atributos de classes não relacionadas que são acessados diretamente ou através de chamadas a métodos de acesso.

8. **ECR (External Calls Rate):** *[4]* Taxa de Chamadas Externas calcula a taxa de chamadas externas realizadas por um método. Essa métrica é uma adaptação da *LAA (Locality of Attribute Accesses)*, que calcula o número de atributos pertencentes a mesma classe que o método em análise, dividido pelo número total de variáveis acessadas.

9. **NCC (Number of Classes Called):** *[4]* Número de Classes Chamadas calcula o número de classes das quais métodos são chamados ou atributos são acessados por um método. Nesse calculo não contamos classes que tenham relação hierárquica com a classe do método em análise. Essa métrica é uma adaptação da *FDP (Foreign Data Providers), que*

calcula o número de classes das quais atributos são acessados. Seu valor varia entre zero e o número total de classes do sistema. Obtemos zero quando o método não utiliza elementos de outras classes.

10. NRA (Number of Reachable Attributes): Número de Atributos Alcançáveis calcula o número de atributos alcançáveis a partir de um método. Obtemos zero quando o método não alcança nenhum atributo de sua classe e o valor máximo quando ele usa direta ou indiretamente todos os atributos de sua classe.

11. MaxNesting (Maximum Nesting Level): *[4]* Nível Máximo de Estruturas Encadeadas calcula o número de caminhos linearmente independentes no método analisado, conhecido como complexidade ciclomática. Seu valor mínimo é 1 e não existe um limite máximo para o seu resultado, mas estamos usando no máximo 7.

12. NP: Número de Parâmetros calcula o número de parâmetros de um método. Seu valor mínimo é zero e não existe um limite máximo para o seu resultado.

13. NRP: Número de Repasses de Parâmetros calcula o número de parâmetros recebidos pelo método em análise e repassados como argumento em chamadas a outras operações pertencentes a sua classe. Calculamos a média de NRP quando precisamos avaliar a média do número de repasses de parâ- metros por método de uma classe.

14. NOA (Number of Attributes): Número de Atributos calcula o número de atributos de uma classe. Obtemos zero quando a classe analisada não possui atributos e estamos atribuindo o máximo 10.

15. AMLOC: Média do Número de Linhas Por Método calcula a média do número de linhas efetivas por método da classe analisada.

Referências Bibliográficas:

[1] MCCONNELL, Steve **Code complete**. Pearson Education, 2004.

[2] MARTIN, Robert C. et al. Código Limpo: Habilidades Práticas do Agile Software.**Rio de Janeiro-RJ: Alta Books**, 2009

[3] HITZ, Markus. Chidamber and Kemerer's metrics suite: a measurement theory perspective. **Software Engineering, IEEE Transactions on**, v. 22, n. 4*, p. 267-271*, 1996. *Citado na pág. 37*

[4] LANZA, Michele; MARINESCU, Radu. **Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems**. Springer Science & Business Media, 2007. *Citado na pág. 4, 35, 36.*

[5] LORENZ, Mark; KIDD, Jeff. **Object-oriented software metrics: a practical guide**. Prentice-Hall, Inc., 1994. *Citado na pág. 37.*

[6] MARINESCU, Radu. Measurement and quality in object-oriented design. In:**Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on**. IEEE, 2005. p. 701-704. *Citado na pág. 35.*

[7] MCCABE, Thomas J. A complexity measure. **Software Engineering, IEEE Transactions on**, n. 4, p. 308-320, 1976. *Citado na pág. 36.*