Marcos Rocha Tavares

Qualidade de Código

O SEI (Software Engineering Institute) definiu o termo métricas de software como um conjunto de medidas de um processo ou produto de software, onde um produto de software pode ser visto como um objeto abstrato que evolui de uma instrução inicial para o sistema de software finalizado.

Então, métricas são utilizadas para estimar um cronograma e custos de desenvolvimento do software e medir a produtividade e qualidade do produto.

É desejável que as métricas possuam as seguintes características (Fenton e Peeger, 1998; Mills, 1988):

• o que a métrica se propõe a medir deve ser claro;

• a métrica deve ser formalmente definida, seu valor deve estar atrelado ao objeto medido, independente de quem (ou qual ferramenta) o obtenha;

• deve ser possível obter seu valor rapidamente e a baixo custo;

• a métrica deve medir efetivamente o proposto por ela;

• pequenas mudanças no software, por exemplo, não podem causar grandes mudanças no valor obtido;

• deve haver formas de mapeamento das métricas para o entendimento do comportamento das entidades analisadas através da manipulação dos números obtidos e

• o resultado da métrica deve ser independente da linguagem de programação utilizada.

1. Começando pelas métricas de tamanho:

LOC (Lines of Code Número de linhas de código) é a medida mais comum para o tamanho de um software. São contadas apenas as linhas executáveis, ou seja, são excluídas linhas em branco e comentários.

AMLOC (Average Method LOC Média do número de linhas de código por método). Essa medida indica se o código está bem distribuído entre os métodos.

MMLOC (Max Method LOC ) número de linhas de código do maior método da classe.

Total Number of Modules or Classes Número total de módulos ou classes é outro indicador de tamanho, que é menos influenciada por linguagens de programação

2. Indicadores estruturais, como descritos a seguir:

NOA (Number of Attributes , Número de atributos) calcula o número de atributos de uma classe.

NOM (Number of Methods , Número de métodos) é usado para medir o tamanho das classes em termos das suas operações implementadas

NPA (Number of Public Attributes , Número de atributos públicos) mede o encapsulamento. Os atributos de uma classe devem servir apenas às funcionalidades da própria classe.

NPM (Number of Public Methods, Número de métodos públicos) representa o tamanho da interface da classe.

ANPM Average Number of Parameters per Method, Média do Número de Parâmetros por Método): calcula a média de parâmetros dos métodos da classe.

DIT (Depth of Inheritance Tree, Profundidade da árvore de herança) é o número de superclasses ou classes ancestrais da classe sendo analisada.

NOC (Number of Children, Número de filhos ): número total de filhos de uma classe (Rosenberg e Hyatt, 1997).

RFC (Response For a Class Respostas para uma classe: número de métodos dentre todos os métodos que podem ser invocados em resposta a uma mensagem enviada por um objeto de uma classe (Sharble e Cohen, 1993).

ACCM(Average Cyclomatic Complexity per Method, Média da Complexidade Ciclomática por método) mede a complexidade do programa (McCabe, 1976).Em outras palavras, se o acoplamento é alto, o software tende a ser menos flexível, mais difícil de se adaptar e modi car e mais difícil de entender.

3. Métricas que medem o acoplamento são explicadas abaixo:

ACC (Aferent Connections per Class, Conexões aferentes de uma classe). Mede a conectividade de uma classe

CBO (Coupling Between Objects, Acoplamento entre objetos) é a recíproca da métrica ACC. Mede quantas classes são utilizadas pela classe analisada

SC (Structural Complexity , Complexidade estrutural). Quanto mais complexo for um software, mais difícil será alterá-lo e evoluí-lo. Acoplamento e coesão foram descritos e discutidos em outros trabalhos como indicadores essenciais de complexidade estrutural (Darcy et al., 2005.

Construção um software não é somente escrever código e vê-lo funcionar, é você saber que aquele código será manutenível e que outras pessoas vão alterá-lo.

Para isso, teste é fundamental! Existe a necessidade de ser responsável por aquilo que escreveu e saber que o sistema tem que continuar funcionando.

Um código limpo deve ser:

• Simples: fácil de entender;

• Direto: direto ao ponto,e objetivo;

• Eficiente: código que faz o que foi proposto;

• Sem duplicidade: não faz o que outra parte do código já faz;

• Elegante: porque é diferente dos outros códigos;

• Feito com cuidado: quem fez teve preocupação em produzir aquele código.

Boas Práticas de Programação

Interessante citar que em grandes equipes de desenvolvimento podem ser definidos padrões para que o código se torne legível para todos programadores e facilitando também o entendimento do código para novos integrantes.

1. Descomplique
2. Testar os comandos da linguagem;
3. Leia e estude cada mensagem de erro emitida pelo compilador;
4. Todo programa deve iniciar com um comentário descrevendo a finalidade do mesmo;
5. Recue o corpo de uma função (endentação), fazendo com que a mesma fique mais legível;
6. Estabeleça então um critério de espaços nos recuos;
7. Declare cada variável em uma linha para facilitar a escrita de comentários;
8. Antes de mais nada, escolha nomes significativos para as variáveis;
9. Declare as variáveis sempre no início das funções;
10. Coloque sempre uma linha em branco entre a declaração de variáveis e os comandos seguintes;
11. Coloque espaços dos dois lados dos operadores aritméticos;
12. Escreva parênteses, ainda que redundantes, para facilitar o entendimento de expressões aritméticas;
13. Do mesmo modo, quebre um comando longo em comandos menores e mais simples;
14. Faça indentação no corpo de uma estrutura;
15. Escreva um comando por linha;
16. separe os comandos longos em várias linhas, quebrando-as em pontos que façam sentido;
17. Consulte as tabelas de precedência dos operadores aritméticos e, em caso de dúvida, escreva parênteses para efetuar as operações da forma pretendida;
18. Use um “pseudocódigo” para “bolar” um programa;
19. Em seguida imprima mensagens de erro em expressões que tenham restrições de valores, tais como divisão por zero, raiz quadrada de número negativo, entrada de dados inválidas, dentre outras;
20. Solicite ao usuário os valores a serem digitados, com mensagens adequadas;
21. “Inicialize” as variáveis ao declará-las, sempre que possível;
22. Controle as repetições com valores inteiros;
23. Evite muitos níveis de indentação, pois estes tornam os programas difíceis de serem entendidos;
24. Usar Try.. Catch
25. Finalmente, teste os programas exaustivamente!

Refatoração

O processo de modificar um sistema de software para aprimorar a estrutura interna do código sem alterar seu comportamento. Isso melhora o desenvolvimento do software sempre sendo aprimorado , e evitando a deterioração , abrindo oportunidade para atualizações de performance e melhoria continua.

Como consequência existe a melhoria no entendimento do código , facilitando a manutenção, utilizando boas práticas de programação e qualificando o código usando métricas melhores de desenvolvimento

Testes automatizados devem servir para identificar se a refatoração não alterou o comportamento do software

Kent Beck, um dos criadores da Programação Extrema, afirma que refatoração deve ser utilizada quando o "código cheirar mal" (do inglês bad smells in code).

Alguns indícios já possuem uma aceitação ampla para promover refatoração:

Código duplicado (duplicated code)

Método longo (long method)

Classe grande (large class)

Lista de parâmetros longa (long parameter list)

Má indentação(Bad Indentation)

As refatorações descritas no livro de Martin Fowler utilizam fortemente conceitos de orientação a objeto. Basta observar algumas:

Extrair Método (Extract Method)

Mover Método (Move Method)

Mover Atributo (Move Field')

Extrair Classe (Extract Class)

Encapsular Atributo (Encapsulate Field)

Renomear Método (Rename Method)

Subir Método (Pull Up Method)

Subir Atributo (Pull Up Field)

Descer Método (Push Down Method)

Descer Atributo (Push Down Field)

Extrair Sub-classe (Extract Subclass)

Extrair Super-classe (Extract Superclass)