

Qualidade de código

Qualidade de Software (2011.0)

Prof. Me. José Ricardo Mello Viana

Conteúdo

1. Introdução
2. Leitura de código: Compreensão da semântica
3. Melhorando a escrita
4. Documentação de código
5. Reaproveitamento de código
6. Programação defensiva
7. Outras técnicas

Introdução

- Programar é um ato individual
- Em equipe, devem contribuir entre si
- Escrever corretamente código ajuda a
 - Redução de defeitos
 - Validação de requisitos
- Geradores de código só resolvem parcialmente
- Ainda é preciso programar
 - Traduzir especificações em código
 - Analisar código escrito por outros
 - Modificar código para remover defeitos
- Manipulação direta de textos escritos em linguagem não natural expressos de maneira não amigável

Introdução

- Pré-requisitos da codificação
 - Linguagens atuais são muito mais complexas
 - Desconhecimento torna o programa mais complicado do que é
 - namespace original {
 - `template <class T> T fun (T i) {`
 - `try { return (T) g((int) i); }`
 - `catch (E) { return (f() == h*()); }`
 - `}`
 - `}`
- Parece intimidador para iniciantes ou não habituado
- Desconhecimento também pode significar esforço desnecessário
 - Aspecto da qualidade de software: economia
 - Escrita e organização de textos
 - Inclusão das informações necessárias
 - Inclusão do código de dispositivos de segurança

Leitura de código: Compreensão da semântica

- Linguagens não foram feitas para serem de fácil leitura
- Contra-exemplos: Cobol e ferramenta Metafor
- Programador deve conhecer melhor a linguagem que usa
- Ex: `while (*pt++ = *ps++); // experiente`
- `while (*ps) { //Não habituado`
 - `*pt = *ps;`
 - `pt++;`
 - `ps++;`
- `}`
- Gasta mais linhas, mais sinais como `;` e `{`
 - Exige mais atenção e tempo de leitura

Leitura de código: Compreensão da semântica

- Estilos de codificação
 - Cada linguagem acaba possuindo um “estilo” de escrita
 - Conhecido e aceito. Regras partem do bom senso.
 - Ex: evitar
 - `for (i = (j > n) ? j : n; i < (k = f(i)[n]); i += (x++, k+=x));`
 - Válido, mas de difícil entendimento
 - Regras simples podem poupar bastante trabalho
 - Evitam erros de compreensão
 - Aprende-se lendo código de outras pessoas
 - Práticas como programação em duplas também são positivas
- Recorrer a revisões também aumenta a padronização

Leitura de código: Compreensão da semântica

- Estilos de codificação

Descrição	Solução
Confusão entre comparação e atribuição	Usar sempre <code>if (5 == a)</code> ao invés de <code>if (a == 5)</code>
Combinar atribuição e comparação	Usar <code>if (NULL != (a = b))</code> em lugar de <code>if (a = b)</code>
Falta de breaks num switch	Usar o auto-completar
Erro de aninhamento de if/else	Usar recuos adequadamente para as chaves

Melhorando a escrita

- Garantir leitura mais fácil do programa trás vantagens
 - Menor probabilidade de o programador perder o controle sobre a complexidade do que está escrevendo
 - Maior facilidade de depurar
 - Melhora do trabalho em equipe
- Razões para escrever código obscuro
 - Falta de disciplina ou treinamento
 - Pressão no ambiente de trabalho
- Como corrigir esses problemas?
 - Veremos dicas variando de identificadores a comentários, até mesmo a própria estética do texto

Melhorando a escrita

- Identificadores
 - Escolhas mnemônicas facilitam o entendimento
 - Ex: s++
 - Associação com tipo char
 - Ex: int x, y, z, x1, x2;
 - É impossível deduzir o propósito de cada variável
 - Mesmo quem escreveu pode ter problemas para entender
 - Debug torna-se mais difícil
 - Bons nomes de identificadores levam a auto-documentação

Melhorando a escrita

- Nomenclatura húngara
 - Tese de doutorado de Charles Simonyi
 - Quando se usa biblioteca MFC ou a API do Windows
 - Prefixa os identificadores para indicar os tipos associados
 - Crítica: Não melhorar a compreensão do funcionamento
 - Ex: `typedef struct _MMIOINFO {`
 - `DWORD dwFlags;`
 - `FOURCC fccIOProc;`
 - `HPSTR pchNext; ...`
 - Prefixo indica o tipo: ch-caractere, p-ponteiro, h-handle
 - `pchNetx` é ponteiro para char
 - Não há informação de funcionamento
 - Alternativa: privilegiar mais a função de cada uma
 - `struct tp_multi_media_info {`
 - `DWORD flags;`
 - `tp_routine *io_routine;`
 - `tp_huge_ptr *buffer`

Melhorando a escrita

- Exemplo de padrão de nomenclatura
 - Não existe padrão universal
 - Cada empresa cria o seu
 - Ex: Ellemtel
 - Como elaborar um padrão
 - Definir um dicionário de prefixos

Prefixo	Significado
arq	Arquivo
c	Caractere
i, j, k, cont	Contador
f, flag	Função booleana

Melhorando a escrita

- Para identificar o escopo
 - G_ global, E_ estática, M_ módulo
 - `_i = E_counter--;`
 - `while (_i < n)`
 - `_i++`
 - Além de variáveis, caracterizar também demais elementos que compõem um programa
 - Tp_ tipo de dado, C_ classe, A_ aspecto, ...
- Pontos negativos
 - Texto fica mais carregado
 - Tempo de adaptação necessário
- Evitar nomes que denotem ação

Melhorando a escrita

- Padrão de codificação web
 - Nomenclatura de formulários e campos

Elemento	Prefixo
Formulário	frm
Texto	txt
TextArea	txa
CheckBox	chk
Hidden	hdn
Password	psw
Botão Radio	rdo
Botão Reset	rst
Botão Submit	sbt
Select	slt

Melhorando a escrita

- Recuos, espaçamento e alinhamento
 - É comum programadores escreverem de forma confusa
 - Algumas dicas
 - Alinhar símbolos de atribuição e vírgulas
 - Separar o texto em parágrafos
 - Substituir comandos if por comparadores
 - Muitos programadores acreditam ser questão de gosto pessoal
 - Trabalhando em equipe deve ser escolhido um padrão

Melhorando a escrita

- Ferramentas apropriadas: editores
 - Possuem recursos que agilizam a escrita
 - Recuos
 - Evidenciar blocos de código
 - Endentação
 - Recuo automático ajuda o programador
 - Geralmente usa-se a tecla TAB
 - Eclipse e Matlab possuem corretor de endentação
 - Code templates
 - Aliviar o problema da repetitividade de código
 - Associação de tecla de atalho ou palavra com trecho de código
 - Reduz o trabalho de digitação
 - Treinamento de novo programador é simplificado

Documentação de código

- Pequenas explicações podem economizar a leitura de dezenas de linhas de código
 - `// Obter o mínimo e o máximo dos bits V e H`
- Comentários podem “amarrar” o código-fonte às especificações
 - `//Laço de contagem de bits`
 - `// Vide docR001.doc pagina 17`
- Depende do estilo de escrita
 - Preguiçoso x Exagerado

Documentação de código

- Ferramenta Doxygen
 - Ao concluir a implementação, normalmente, há mais artefatos que o previsto
 - A documentação construída contribui para o entendimento completo
 - Há ferramentas que dão suporte a documentação
 - Inclusive mostrando relações de hierarquia
 - Doxygen: gratuita, analisa o código e gera relatórios (C, C++, Java)
 - Diagramas de herança
 - Declarações de estruturas de dados
 - Listas alfabéticas de sub-rotinas, módulos, variáveis, etc
 - Pode gerar HTML, PDF ou RTF
 - Extrai os comentários contidos no código

Documentação de código

- Formatos de comentários
 - Comentários com formato padrão são usados para
 - Documentar classes, variáveis globais ou sub-rotinas
 - Exemplo pag 320
 - Inclusão nos comentários de:
 - Efeitos colaterais: sub-rotina altera o estado global do sistema
 - Casos não tratados: situações que o código não estava preparado
 - Campos podem ser omitidos se ferramenta for usada

Reaproveitamento de código

- Problemas já resolvidos não devem ser resolvidos novamente
- Numa equipe, diversas pessoas usam funções comuns
 - Validar CPF e data
- Perda de tempo e propagação de defeitos pode acontecer
- Pode-se usar um repositório de funções
- Reuso de trechos de código
 - Copiar código pronto e adaptá-lo
 - Deve ser feito seguindo toda a lógica da engenharia de software
 - Verificar os requisitos sendo tratados pelo código copiado

Reaproveitamento de código

- Comunidades de desenvolvedores
 - A internet é uma fonte de possibilidades
 - Tutoriais
 - Programas completos
 - Bibliotecas de funções
 - Ponto alto: comunidades
 - Fóruns de discussão
 - Material gratuito fornecido pelos próprios leitores
 - Projetos inteiros
 - Artigos técnicos e tutoriais

Reaproveitamento de código

- Bibliotecas
 - Permite ao desenvolvedor se concentrar no que sua aplicação deve prover
 - Há bibliotecas para diversas áreas, como
 - Compressão e criptografia
 - Construção de interfaces
 - Comunicação em rede
 - Cálculo numérico
 - Manipulação gráfica 2D e 3D
 - Existem comerciais e gratuitas, com código disponível ou não
 - Ao escolher uma, o desenvolvedor deve analisar
 - Documentação: Boas bibliotecas contem descrição de uso clara acompanhada
 - Histórico de revisão: Código mais antigo é código testado mais vezes
 - Fonte: quando disponível, deve ser analisado

Reaproveitamento de código

- STL
 - Biblioteca padrão de algoritmos e estruturas de dados para C++
 - Infelizmente, muitos a desconhecem
 - Perdem tempo fazendo código que já existe
 - Exemplo página 324
 - Principal conceito STL: container
 - Classe que encapsula uma estrutura de dados
 - A eles é possível aplicar algoritmos, também fornecidos pela STL
 - É necessária certa fluência em C++ e conhecimento de templates

Programação defensiva

- Atitude de desconfiança a respeito de tudo que possa dar errado
- Traduz-se em aumentar o controle sobre o programa
 - Inclusão de mais verificações no código
- Uma série de práticas podem ser adotadas
- Entrada de dados
 - Um programa não deveria aceitar entradas erradas
 - Crítica via usuário
 - Dados fornecidos de rotina para rotina
 - Consistência das estruturas de dados
 - É possível prever com tratar erros

Programação defensiva

- Compilação condicional e assertivas
 - Permite criar diferentes versões do programa para diferentes plataformas
 - #ifdef UNIX
 - #elif WINDOWS
 - #endif
 - Também pode ser usada para criar versões de testes e para depuração
- Exceções
 - Condição inesperada que causaria uma falha
 - Solução mais limpa que testar através de if
 - Caso ocorra algum problema, o programa será avisado

Programação defensiva

- Estilo de codificação
 - Programação defensiva relaciona-se também com a forma com que o código é escrito
 - Ao usar os recursos da linguagem escolhida o programador deve estar absolutamente convicto do que está fazendo
 - Ex: Ao invés de escrever
 - `double a, b;`
 - `if (a == b)`
 - Prefira
 - `if (abs(a - b) < TOLERANCIA)`
 - Conselho: habilitar todos os warnings do compilador

Outras técnicas

- CleanRoom
 - Sala limpa = ambiente estritamente controlado
 - Orientada a equipes e baseia-se em controles estatísticos
 - O processo de engenharia de software deve ser baseado em fundamentos matemáticos, não em tentativa e erro
 - Características
 - Fazer certo da primeira vez
 - Desenvolvimento incremental, aprovando cada etapa
 - Especificação formal dos programas
 - Teste de software segue princípios estatísticos
 - Não foi largamente usada, apesar dos bons resultados
 - Crença em ser teórica demais e baseada em muita matemática
 - Mudança radical no desenvolvimento, menos testes e mais controle estatístico
 - Imaturidade dos processos de desenvolvimento das organizações

Outras técnicas

- Programação por contrato
 - Baseia-se na noção de assertivas
 - Em certos pontos do programa são estabelecidas condições que, se violadas, indicam a ocorrência de falhas
 - Baseado na lógica de Floyd-Hoare
 - Dois tipos de condições
 - Pré-condições que devem ser satisfeitas para que o trecho seguinte seja executado corretamente
 - Pós-condições que devem ser satisfeitas ao final da execução do trecho considerado
 - Pode-se incluir ainda condições invariantes