Aula 03

Programação Modular

Programação II, 2020-2021

v2.10, 26-02-2020

DETI, Universidade de Aveiro

Sumário

Aspectos da qualidade relacionados com o desempenho do software

- Correcção: o software efectua as suas funções exactamente como definido nas suas especificações;
- Robustez: o software "funciona" em situações fora das suas especificações;
- Fiabilidade: o software é correcto e robusto;
- Eficiência: o software utiliza o mínimo de recursos de hardware necessários (directamente relacionada com a complexidade algorítmica):
  - CPU: tempo de execução;
  - RAM: memória gasta.

Aspectos da qualidade relacionados com o processo de desenvolvimento

- Extensibilidade: o software é fácil de adaptar a mudanças de especificações;
- Reutilização: o software pode ser utilizado total, ou parcialmente, para novas aplicações;
- Legibilidade - facilidade de leitura e compreensão de software;
- Verificabilidade - facilidade com que o mesmo pode ser testado;

Aspectos da qualidade do software

- Todos estes aspectos são relevantes, mas não têm todos a mesma importância.
  - O mais importante de todos é a correcção. Se um programa/função/módulo não faz o que é suposto fazer, é irrelevante que seja robusto ou eficiente (por exemplo).
  - O segundo mais importante é a robustez.
  - Na construção de programas devem usar-se técnicas que maximizem estes dois factores de qualidade.

Modularidade: introdução

Os programas vão crescendo em tamanho e complexidade levantando novas questões:

- Torna-se cada vez mais importante não só o seu funcionamento externo mas também a forma como é construído;
- Pode existir a necessidade de ter várias pessoas a trabalhar simultaneamente no programa, pelo que uma comunicação fácil entre programadores através do próprio código passa a ser cada vez mais determinante;
- O número potencial de erros tende a aumentar obrigando a técnicas que facilitem a sua detecção e atempada correcção (e que evitem um crescimento exponencial da complexidade no seu tratamento);
- É necessário facilitar a manutenção e eventuais futuras evoluções do programa (extensibilidade).

Modularidade: conceito e critérios

Módulos

Excertos de programas (blocos) independentes com os quais se podem construir novos programas.

Dizemos que um bloco de um programa é modular se for:

1. Facilmente separável de outros blocos;
2. Facilmente combinável com outros blocos;
3. Fácil de ser compreendido isoladamente;
4. Continuidade: pequenas modificações num módulo apenas o afectam a ele ou eventualmente aos seus clientes directos;
5. Auto-protegido: dados internos protegidos contra utilizações abusivas.

Modularidade: vantagens

- Cada módulo pode ser desenvolvido, analisado e testado de forma independente:
  - Pode ser da responsabilidade de entidades (pessoas) distintas.
- Mais fácil de maximizar a correcção e a robustez;
- Reduz a complexidade do programa global:
  - Implementação de mecanismos de abstracção para facilitar tarefas.
- Facilita a reutilização de código:
  - Ao desenvolvermos um módulo especializado numa tarefa/funcionalidade, podemos facilmente reutilizá-lo noutro programa com as mesmas necessidades.

Evolução histórica

1. Programa monolítico
2. Abstracção algorítmica
3. Registos

} Programação I

4. Ficheiros como unidades de suporte à modularidade
   - (*a*) Prog. Principal + Módulo Tipo Dados + Módulo Funções
   - (*b*) Prog. Principal + Módulo Tipo Dados e Operações Associadas
5. Contexto de Existência de Objecto
6. Encapsulamento: Objectos e Tipos de Dados Abstractos

Programação modular – passo 1

PROGRAMA $\Leftrightarrow$ FUNÇÃO `main`

Programa monolítico!

O único “módulo” é o próprio programa

Programação modular – passo 2

PROGRAMA $\Leftrightarrow$ `main` + funções (no mesmo ficheiro)

ABSTRACÇÃO ALGORÍTMICA!

Funções podem ser utilizadas como módulos internos ao programa

Abstracção algorítmica!

Criação de Funções (métodos):

- Encapsulamento de uma sequência de instruções dentro de um módulo funcional;
- A compreensão desse módulo reduz-se à compreensão da sua utilização (e não da sua implementação);
- Tal como os programas, as funções podem ter parâmetros de entrada e de saída;
- Permite a sua fácil reutilização dentro do programa sem a necessidade de replicar a sua implementação;
- Podemos associar-lhes especificações formais de correcção através de pré-condições e pós-condições.

Problemas:

- O facto de serem unidades de código que são compiladas conjuntamente com o programa principal limita a sua reutilização (só utilizando "*copy & paste*");
- A representação interna dos dados manipulados pelas funções está propagada em todo o lado.

Programação modular – passo 3

REGISTOS

PROGRAMA $\Leftrightarrow$ `main` + funções + registos

1ª ABSTRACÇÃO DE DADOS

Representação de dados dos módulos deixa de estar directamente exposta aos clientes

Registos

Criação de novos tipos de dados (registos):

- Primeira abstracção de dados;
- Encapsulamento de um conjunto de tipos de dados dentro de um novo tipo de dados;
- O registo pode ser compreendido pelo seu todo (e não somente pelos tipos de dados sobre os quais é construído);
- Podemos lhes associar uma especificação formal de correcção através de um invariante (em `Java` não há suporte em tempo de execução para este formalismo).

Problemas:

- Dependem fortemente das funções que os manipulam;
- O facto de serem unidades de código que são compilados conjuntamente com o programa principal limita a sua reutilização (só utilizando "*copy & paste*").

Programação modular – passo 4(a)

MÓDULOS REUTILIZÁVEIS DE FUNÇÕES
MÓDULOS REUTILIZÁVEIS DE REGISTOS

PROGRAMA $\Leftrightarrow$ `main` + módulos de funções + módulos de registos

Módulos separados em diferentes ficheiros
Funções podem ser (re)utilizadas noutros programas

Representação de dados dos módulos directamente exposta aos clientes

Criação de novos módulos

- Unidades autónomas (ficheiros) contendo:
  - Tipo de Dados;
  - Funções.
- Podem ser invocados do exterior;
- Podem ser compilados isoladamente e "ligados" (*linked*) a outros programas que deles necessitem facilitando assim a reutilização de código;
- Programação Modular orientada a funções; (e.g.: biblioteca `Math`).

Questão:

- Se as operações estão associadas a um tipo de dados, faz sentido manter DADOS e OPERAÇÕES em ficheiros diferentes?

Programação modular – passo 4(b)

JUNÇÃO REGISTOS COM MÓDULOS DE FUNÇÕES

PROTECÇÃO E ABSTRACÇÃO DE DADOS

PROGRAMA $\Leftrightarrow$ `main` + módulos de registos/funções

Registo é passado como argumento das funções do próprio módulo

Programação modular – passo 4(b)

- Já vimos que a definição de novos tipos de dados de pouco serve se não forem definidas operações sobre eles:
  - Manipulação de dados;
  - Comparação, Atribuição, ...
  - Operações algébricas;
  - etc.
- Então: "Qual o interesse de termos os Dados e Funções Associadas em módulos diferentes?"
- Surgiu então um novo conceito:
  Módulo: Novo Tipo de Dados + Conjunto de Operações Associadas

Programação modular – passo 5

Contexto de Existência de Objecto!

PROGRAMA $\Leftrightarrow$ `main` + objectos

As funções do módulo (métodos) e a respectiva estrutura de dados são indissociáveis.

Os métodos estáticos que recebiam o registo como argumento passam a métodos de objecto funcionando no contexto do objecto.

Programação modular – passo 6

OBJECTOS!

PROGRAMA $\Leftrightarrow$ `main` + objectos

Nenhum atributo deverá ser tornado público (ao contrário dos registos)

Objectos

Módulos absolutamente independentes:

- Cada módulo só deve aceder a dados locais;
- A interacção com o exterior deve ser efectuada através de funções do próprio módulo (interface);
- Obriga a utilizar mecanismos de protecção para "esconder" os dados do mundo exterior:
  `public / private / protected`

Um novo conceito:

- Encapsulamento (*Information Hiding*)!

Information hiding

A informação contida num módulo está inacessível a outros módulos que não têm necessidade dessa informação.