Paradigmas Fundamentales de Programación

Diseño de programas en pequeño

Juan Francisco Díaz Frias

Maestría en Ingeniería, Énfasis en Ingeniería de Sistemas y Computación
Escuela de Ingeniería de Sistemas y Computación,
home page: http://eisc.univalle.edu.co
Universidad del Valle - Cali, Colombia

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

Metodología de diseño

Programar en pequeño y en grande

- Los programas pequeños: escritos por una persona en un tiempo corto.
- Los programas grandes: escritos por más de una persona en un tiempo largo.
- Usar una metodología de diseño que sea mezcla de creatividad y pensamiento riguroso.

Metodología recomendada (flexible)

- Especificación informal: Escritura precisa pero en lenguaje natural, de las entradas y salidas y cómo se relacionan las segundas con las primeras.
- Ejemplos: lo que el programa hace en casos particulares. Los ejemplos deben "presionar" el programa: usarlo en condiciones límites y en las formas más inesperadas que podamos imaginar.

Metodología recomendada (cont.)

Metodología de diseño

Programar en pequeño y en grande

- Los programas pequeños: escritos por una persona en un tiempo corto.
- Los programas grandes: escritos por más de una persona en un tiempo largo.
- Usar una metodología de diseño que sea mezcla de creatividad y pensamiento riguroso.

Metodología recomendada (flexible)

- Especificación informal: Escritura precisa pero en lenguaje natural, de las entradas y salidas y cómo se relacionan las segundas con las primeras.
- Ejemplos: lo que el programa hace en casos particulares. Los ejemplos deben “presionar” el programa: usarlo en condiciones límites y en las formas más inesperadas que podamos imaginar.

Metodología de diseño

Programar en pequeño y en grande

- Los programas pequeños: escritos por una persona en un tiempo corto.
- Los programas grandes: escritos por más de una persona en un tiempo largo.
- Usar una metodología de diseño que sea mezcla de creatividad y pensamiento riguroso.

Metodología recomendada (flexible)

- Especificación informal: Escritura precisa pero en lenguaje natural, de las entradas y salidas y cómo se relacionan las segundas con las primeras.
- Ejemplos: lo que el programa hace en casos particulares. Los ejemplos deben “presionar” el programa: usarlo en condiciones límites y en las formas más inesperadas que podamos imaginar.

Metodología recomendada (cont.)

- Exploración: utilizar la interfaz interactiva para experimentar con fragmentos de programa. Visión más clara de lo que debe ser la estructura del programa.
- Estructura y codificación: un bosquejo aproximado de las operaciones que se necesitan y de cómo lograr que estas operaciones se comporten bien juntas. Operaciones sencillas y agrupadas en módulos.
- Comprobación y razonamiento: verificar que el programa hace lo correcto. Casos de prueba, incluyendo los ejemplos.
- También podemos razonar sobre el programa y su complejidad.

Metodología de diseño

Programar en pequeño y en grande

- Los programas pequeños: escritos por una persona en un tiempo corto.
- Los programas grandes: escritos por más de una persona en un tiempo largo.
- Usar una metodología de diseño que sea mezcla de creatividad y pensamiento riguroso.

Metodología recomendada (flexible)

- Especificación informal: Escritura precisa pero en lenguaje natural, de las entradas y salidas y cómo se relacionan las segundas con las primeras.
- Ejemplos: lo que el programa hace en casos particulares. Los ejemplos deben “presionar” el programa: usarlo en condiciones límites y en las formas más inesperadas que podamos imaginar.

Metodología recomendada (cont.)

- Exploración: utilizar la interfaz interactiva para experimentar con fragmentos de programa. Visión más clara de lo que debe ser la estructura del programa.
- Estructura y codificación: un bosquejo aproximado de las operaciones que se necesitan y de cómo lograr que estas operaciones se comporten bien juntas. Operaciones sencillas y agrupadas en módulos.
- Comprobación y razonamiento: verificar que el programa hace lo correcto. Casos de prueba, incluyendo los ejemplos.
  También podemos razonar sobre el programa y su complejidad.
- Juicio de calidad: ¿resuelve el diseño el problema planteado, es correcto el diseño, es eficiente, se puede mantener, se puede extender, es sencillo?

Metodología de diseño

Programar en pequeño y en grande

- Los programas pequeños: escritos por una persona en un tiempo corto.
- Los programas grandes: escritos por más de una persona en un tiempo largo.
- Usar una metodología de diseño que sea mezcla de creatividad y pensamiento riguroso.

Metodología recomendada (flexible)

- Especificación informal: Escritura precisa pero en lenguaje natural, de las entradas y salidas y cómo se relacionan las segundas con las primeras.
- Ejemplos: lo que el programa hace en casos particulares. Los ejemplos deben “presionar” el programa: usarlo en condiciones límites y en las formas más inesperadas que podamos imaginar.

Metodología recomendada (cont.)

- Exploración: utilizar la interfaz interactiva para experimentar con fragmentos de programa. Visión más clara de lo que debe ser la estructura del programa.
- Estructura y codificación: un bosquejo aproximado de las operaciones que se necesitan y de cómo lograr que estas operaciones se comporten bien juntas. Operaciones sencillas y agrupadas en módulos.
- Comprobación y razonamiento: verificar que el programa hace lo correcto. Casos de prueba, incluyendo los ejemplos.
  También podemos razonar sobre el programa y su complejidad.
- Juicio de calidad: ¿resuelve el diseño el problema planteado, es correcto el diseño, es eficiente, se puede mantener, se puede extender, es sencillo?

Metodología de diseño

Programar en pequeño y en grande

- Los programas pequeños: escritos por una persona en un tiempo corto.
- Los programas grandes: escritos por más de una persona en un tiempo largo.
- Usar una metodología de diseño que sea mezcla de creatividad y pensamiento riguroso.

Metodología recomendada (flexible)

- Especificación informal: Escritura precisa pero en lenguaje natural, de las entradas y salidas y cómo se relacionan las segundas con las primeras.
- Ejemplos: lo que el programa hace en casos particulares. Los ejemplos deben “presionar” el programa: usarlo en condiciones límites y en las formas más inesperadas que podamos imaginar.

Metodología recomendada (cont.)

- Exploración: utilizar la interfaz interactiva para experimentar con fragmentos de programa. Visión más clara de lo que debe ser la estructura del programa.
- Estructura y codificación: un bosquejo aproximado de las operaciones que se necesitan y de cómo lograr que estas operaciones se comporten bien juntas. Operaciones sencillas y agrupadas en módulos.
- Comprobación y razonamiento: verificar que el programa hace lo correcto. Casos de prueba, incluyendo los ejemplos.
  También podemos razonar sobre el programa y su complejidad.
- Juicio de calidad: ¿resuelve el diseño el problema planteado, es correcto el diseño, es eficiente, se puede mantener, se puede extender, es sencillo?

Metodología de diseño

Programar en pequeño y en grande

- Los programas pequeños: escritos por una persona en un tiempo corto.
- Los programas grandes: escritos por más de una persona en un tiempo largo.
- Usar una metodología de diseño que sea mezcla de creatividad y pensamiento riguroso.

Metodología recomendada (flexible)

- Especificación informal: Escritura precisa pero en lenguaje natural, de las entradas y salidas y cómo se relacionan las segundas con las primeras.
- Ejemplos: lo que el programa hace en casos particulares. Los ejemplos deben “presionar” el programa: usarlo en condiciones límites y en las formas más inesperadas que podamos imaginar.

Metodología recomendada (cont.)

- Exploración: utilizar la interfaz interactiva para experimentar con fragmentos de programa. Visión más clara de lo que debe ser la estructura del programa.
- Estructura y codificación: un bosquejo aproximado de las operaciones que se necesitan y de cómo lograr que estas operaciones se comporten bien juntas.
  Operaciones sencillas y agrupadas en módulos.
- Comprobación y razonamiento: verificar que el programa hace lo correcto. Casos de prueba, incluyendo los ejemplos.
  También podemos razonar sobre el programa y su complejidad.
- Juicio de calidad: ¿resuelve el diseño el problema planteado, es correcto el diseño, es eficiente, se puede mantener, se puede extender, es sencillo?

Componentes de software

¿Cómo organizar un programa?

- Dividir el programa en unidades lógicas, cada una de las cuales implementa un conjunto de operaciones relacionadas de alguna manera.
- Cada unidad lógica consta de dos partes, una interfaz y una implementación.
- La interfaz es lo visible.
- Una unidad lógica puede usar otras como parte de su implementación.
- Un programa es un grafo dirigido de unidades lógicas.
- Estas unidades lógicas son llamadas popularmente “módulos” o “componentes”: acá los definiremos precisamente.

Módulo

Agrupa un conjunto de operaciones relacionadas en una entidad que posee una interfaz y una implementación.

- La interfaz es un registro que agrupa entidades relacionadas del lenguaje
- La implementación del módulo es un conjunto de entidades de lenguaje a las que se puede acceder por medio de la interfaz, pero están ocultas.

Especificación de módulo

- Tipo de plantilla que crea un módulo cada vez que se instancia: functors.
- Un functor es una función cuyos argumentos son los módulos que necesita y cuyo resultado es un módulo nuevo.
- Una aplicación autónoma es un functor que se evalúa cuando el programa empieza.

Componentes de software

¿Cómo organizar un programa?

- Dividir el programa en unidades lógicas, cada una de las cuales implementa un conjunto de operaciones relacionadas de alguna manera.
- Cada unidad lógica consta de dos partes, una interfaz y una implementación.
- La interfaz es lo visible.
- Una unidad lógica puede usar otras como parte de su implementación.
- Un programa es un grafo dirigido de unidades lógicas.
- Estas unidades lógicas son llamadas popularmente “módulos” o “componentes”: acá los definiremos precisamente.

Módulo

Agrupa un conjunto de operaciones relacionadas en una entidad que posee una interfaz y una implementación.

- La interfaz es un registro que agrupa entidades relacionadas del lenguaje
- La implementación del módulo es un conjunto de entidades de lenguaje a las que se puede acceder por medio de la interfaz, pero están ocultas.

Especificación de módulo

- Tipo de plantilla que crea un módulo cada vez que se instancia: functors.
- Un functor es una función cuyos argumentos son los módulos que necesita y cuyo resultado es un módulo nuevo.
- Una aplicación autónoma es un functor que se evalúa cuando el programa empieza.

Componentes de software

¿Cómo organizar un programa?

- Dividir el programa en unidades lógicas, cada una de las cuales implementa un conjunto de operaciones relacionadas de alguna manera.
- Cada unidad lógica consta de dos partes, una interfaz y una implementación.
- La interfaz es lo visible.
- Una unidad lógica puede usar otras como parte de su implementación.
- Un programa es un grafo dirigido de unidades lógicas.
- Estas unidades lógicas son llamadas popularmente “módulos” o “componentes”: acá los definiremos precisamente.

Módulo

Agrupa un conjunto de operaciones relacionadas en una entidad que posee una interfaz y una implementación.

- La interfaz es un registro que agrupa entidades relacionadas del lenguaje
- La implementación del módulo es un conjunto de entidades de lenguaje a las que se puede acceder por medio de la interfaz, pero están ocultas.

Especificación de módulo

- Tipo de plantilla que crea un módulo cada vez que se instancia: functors.
- Un functor es una función cuyos argumentos son los módulos que necesita y cuyo resultado es un módulo nuevo.
- Una aplicación autónoma es un functor que se evalúa cuando el programa empieza.

Implementando módulos y functors (1)

Ejemplo de módulo

```
declare MyList in
local
   proc {Append ... } ... end
   proc {MergeSort ...} ... end
   proc {Sort ... } ... {MergeSort ...} ... end
   proc {Member ...} ... end
in
   MyList='export'(append: Append
                   sort: Sort
                   member: Member
                   ...)
end
```

- Un módulo es un registro, y su interfaz se accede a través de los campos del registro.
- `MergeSort` es inalcanzable; los otros procedimientos se alcanzan a través de los campos de `MyList`.

Implementando módulos y functors (2)

Una especificación o componente

```
fun {MyListFunctor}
   proc {Append ... }  ... end
   proc {MergeSort ...} ... end
   proc {Sort ... } ... {MergeSort ...} ... end
   proc {Member ...} ... end
in
   'export'(append: Append
            sort: Sort
            member: Member
            ...)
end
```

- Un functor es un valor:se puede evaluar en tiempo de ejecución, puede tener referencias externas, se puede almacenar en un archivo, es posible manipularlos con el lenguaje en formas muy sofisticadas..

Implementando módulos y functors (3)

Soporte lingüístico

```
functor
export
   append:Append
   sort:Sort
   member:Member
   ...
define
   proc {Append ... }  ... end
   proc {MergeSort ...} ... end
   proc {Sort ... } ... {MergeSort ...} ... end
   proc {Member ...} ... end
end
```

- La declaración entre **define** y **end** declara variables implíctamente.

Implementando módulos y functors (4)

Sintáxis general de un functor

⟨declaración⟩ ::= **functor** ⟨variable⟩
[**import** { ⟨variable⟩ [**at** ⟨átomo⟩]
| ⟨variable⟩ **'('** { (⟨átomo⟩ | ⟨ent⟩) [**':'** ⟨variable⟩] }+ **')'**
}+]
[**export** { [(⟨átomo⟩ | ⟨ent⟩) **':'**] ⟨variable⟩ }+]
define { ⟨parteDeclaración⟩ }+ [**in** ⟨declaración⟩] **end**
| ...

El mundo real: requerimientos no declarativos

Para conectar un programa declarativo al mundo real, se necesitan algunas operaciones no declarativas:

- De entrada/salida : módulos `File` (para entrada/salida de archivos de texto) y `Pickle` (para entrada/salida de cualesquier tipo de valores).
- Interfaces gráficas de usuario: módulo `QTk`.
- Compilación de aplicaciones autónomas: **`functor`**.