Mantenimiento del Software

S7

Francisco Ruiz, Macario Polo

Grupo Alarcos

Dep. de Informática

ESCUELA SUPERIOR DE INFORMÁTICA

UNIVERSIDAD DE CASTILLA-LA MANCHA

http://alarcos.inf-cr.uclm.es/doc/mso/

Ciudad Real, 2000/2001

Índice - Sesión 7

- Ingeniería inversa de programas
- Ingeniería inversa de interfaces de usuario
- Algunas herramientas para ingeniería inversa.

Ingeniería inversa de programas

- ¿Qué solemos hacer?
  - Buscamos el programa principal
  - Despreciamos inicializaciones de variables, etc.
  - Inspeccionamos la primera rutina llamada, y la examinamos si es importante
  - Inspeccionamos las rutinas llamadas por la primera rutina del programa principal, y examinamos aquéllas que nos parecen importantes
  - ...

Ingeniería inversa de programas

- Recopilamos esas rutinas “importantes”, que se llaman *componentes funcionales*
- Asignamos significado a cada componente funcional (c.f.):
  - Explicamos qué hace cada c.f. en el conjunto del sistema
  - Explicamos qué hace el sistema a partir de los diferentes cc.ff.

Ingeniería inversa de programas

- Los módulos suelen estar ocupados por c.f.
- Suele haber cc.ff. cerca de grandes zonas de comentarios (líneas de asteriscos, p. ej.)
- Los identificadores suelen ser largos y formados por palabras entendibles

- Un componente es *funcional* cuando su ausencia...
  - ...impide seriamente el funcionamiento de la aplicación
  - ...dificulta la legibilidad del código
  - ...impide la comprensión de todo o de otro c.f.
  - ...hace caer a niveles muy bajos la calidad, fiabilidad, mantenibilidad, etc.

Reengineering Object Oriented Code (Fanta y Rajlich, 1998)

- Reestructuración de una aplicación CAD en C++
- 120.000 LDC, 200 ficheros, 80 clases, 50 funciones globales
- Código y datos “mal colocados”
- Clones (compilador utilizado no soportaba plantillas)
- Degradación progresiva
- Disminución de la mantenibilidad
- Incremento de la dificultad de modificar el código

- Herramientas para:
  - Inserción de funciones: coloca una función dentro de una clase
  - Encapsulación de código en funciones
  - Expulsión de funciones: sacar funciones de una clase

Inserción de funciones

Inserción de una función en una clase y convertirla en pública

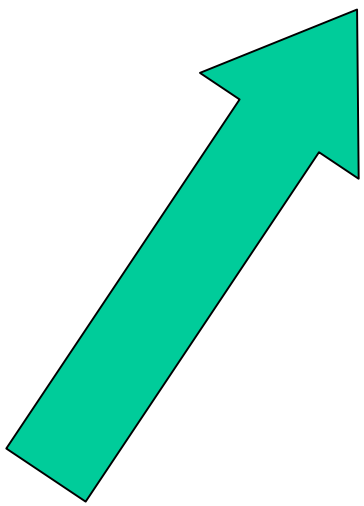
Inserción de funciones

- Pasos:
  - Insertar la cabecera en la declaración de la clase.
  - Actualizar cuerpo de la función (acceso directo a los miembros de la clase).
  - Actualizar cabecera de la función (preceder con el identificador de clase; eliminar el parámetro).
  - Actualizar llamadas a la función (añadir instancias de clase sobre las que ejecutar las llamadas).
  - Eliminar declaraciones de la función.

Ejemplo

```
class A
{
public:
      int i;
protected:
      char c;
};
        ...

int f(A& a)
{
      a.i=4;
      ...
}
```



```
class A
{
public:
      int i;
      int f();
protected:
      char c;
};
        ...

int A :: f() {
      this->i=4;
      ...
};
```

Encapsulación de código en funciones

Encapsula una secuencia de código en funciones.

Encapsulación de código en funciones

- El usuario selecciona un bloque de código y la herramienta decide si éste es sintácticamente completo.
- En caso afirmativo, coloca el código en una nueva función y reemplaza el bloque original por una llamada.
- La función creada puede pasar a ser miembro de una clase.

Encapsulación de código en funciones

- Tratamiento de las variables del bloque:
  - Genera una vble. local por cada variable del bloque que no posea información para el resto del bloque.
  - Las vbles. globales permanecen como globales.
  - Se pasan por valor las vbles. locales que se usan posteriormente sin haber cambiado su valor.
  - El resto, se pasan por referencia.

```
void f(char c)
{
   int i, count, len;
   char str[max;

   cin >> str;
   len=strlen(str);

   count=0;
   for (i=0; i<=len; i++)
        if (str[i]==c) {
            count++;
            str[i]='\n';
        }

   cout << str << count << "\n";
}
```

```
void f(char c)
{
   int i, count, len;
   char str[max;

   cin >> str;
   len=strlen(str);
   newfun(count, len, str, c);
   cout << str << count << "\n";
}

newfun(int &count, int len, char
   *str, char c)
{
   int i;
   count=0;
   for (i=0; i<=len; i++)
        if (str[i]==c) {
            count++;
            str[i]='\n';
        }
}
```

Expulsión de funciones

Saca una función miembro de una clase y la convierte en una función aislada. La clase será pasada a la función como parámetro.

Expulsión de funciones

- Pasos:
  - Se quita la cabecera de la función de la declaración de clase.
  - Se generan, en caso necesario, métodos públicos para acceder a los miembros privados de la clase a los que accedía la función.
  - Se actualiza la cabecera de la función.
  - Se actualiza el cuerpo de la función.
  - Se actualizan las llamadas.
  - Se incluye la declaración temprana de la función.

Generating Objects from C Code (Taschwer, 1999)

- Fases:
  - Preprocesamiento.
  - Construcción de un árbol de sintaxis abstracta (ASA)
  - Generación de clases
  - Asignación de funciones a clases
  - Conversión de las funciones a métodos

Construcción del A.S.A.

- Se genera un árbol completo, que representa todos los ficheros del programa
- Requisitos:
  - Persistencia
  - Facilitar el acceso a identificadores y nombres de tipo (*typedef*)
  - Sustituir nombres de tipo por declaraciones de tipos
  - Preservar el orden de inclusión de los ficheros.

Generación de clases

- Reescritura de estructuras como clases:

```
struct point
 {
     int x, y;
     struct point *next;
};
```

```
class point
{
public:
     int x, y;
     struct point *next;
};
```

Asignación de funciones a clases

- Criterios:
  - Si hay exactamente un parámetro de clase A y el valor devuelto no es de ninguna clase...

    ...convertir la función en un método de A.

  - Si no hay parámetros y el tipo de la función es la clase R...

    ...convertir la función en un método de R.

  - Si hay exactamente un parámetro de clase A y el valor devuelto es de clase R...

    ...convertir a un método de A o R, dependiendo de la última clase declarada

  - Si hay parámetros de dos clases distintas: no hace nada

```
struct Persona
{
   char nombre[10];
   int edad;
};

int ShowEdad(Persona x)
{
   return x.edad;
}
```

```
class Persona
{
public:
   char nombre[10];
   int edad;
   int ShowEdad();
};

int Persona :: ShowEdad()
{
   return this->edad;
}
```

```
struct Persona
{
   char nombre[10];
   int edad;
};

Persona Crea()
{
   Persona x;
   strcpy(x.nombre, "Ana");
   x.edad = 20;
   return x;
}
```

```
class Persona
{
public:
   char nombre[10];
   int edad;
   Persona Crea();
};

Persona Persona :: Crea()
{
   strcpy(nombre, "Ana");
   edad = 20;
   return this;
}
```

Conversión de las funciones a métodos

- Las funciones asignadas en la fase anterior se transforman en métodos de la siguiente forma:
  - Si la función (que ya es un método) se asignó a la clase A porque ésta aparecía en la lista de parámetros, el método se declara como no estático.
  - Si la función (que ya es un método) se asignó a la clase A porque ésta es el tipo devuelto, se declara como estático.

Conversión de las funciones a métodos

- Todas las funciones transformadas en métodos se declaran como públicas.

- Si la clase aparece en la lista de parámetros de la antigua función, se elimina en la lista de parámetros del método (excepto si el parámetro es pasado por referencia) y se sustituyen las referencias al parámetro por construcciones *this*.

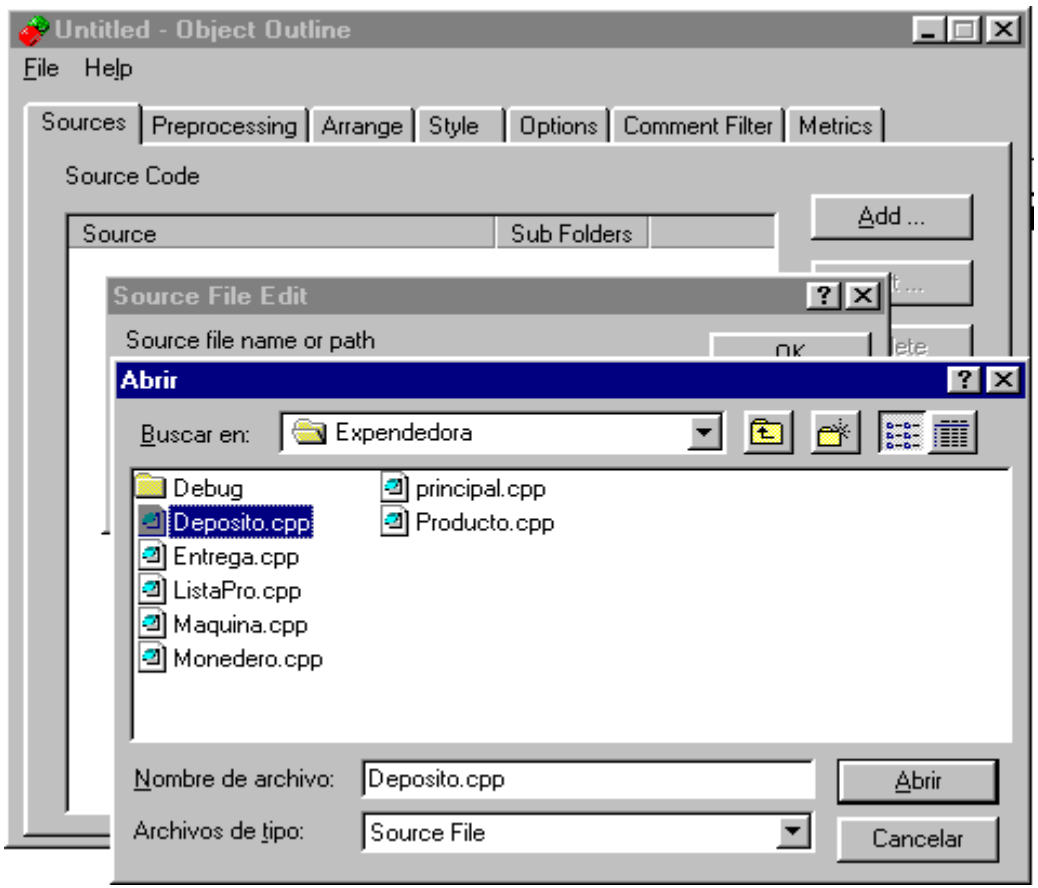
Para entretenerse...

- Gramática para C:
  ftp://ftp.uu.net/usenet/net.sources/ansi.c.grammar.Z

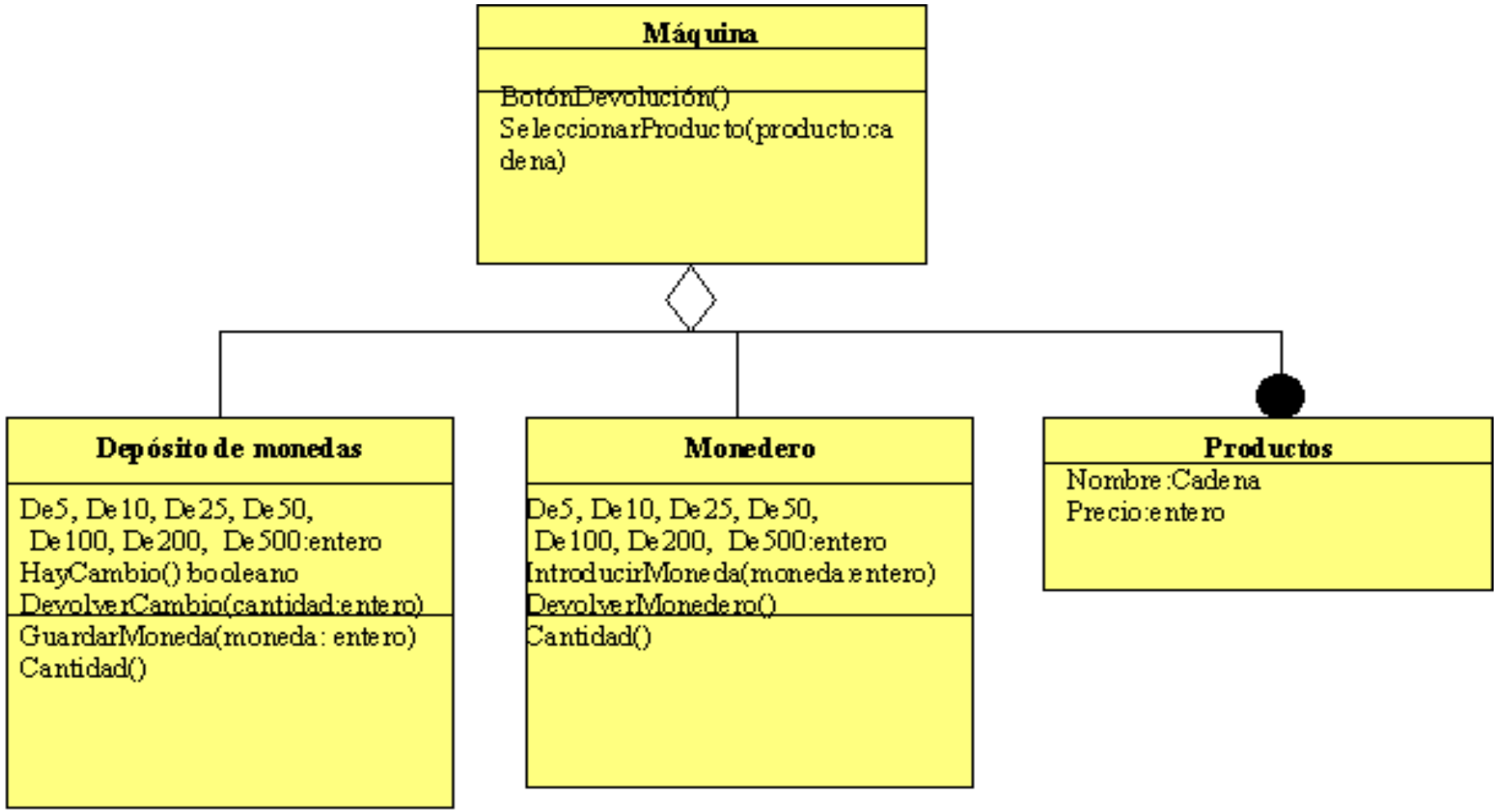
- Gramática para C++:
  http://www.empathy.com/pccts/roskind.html

Object outline

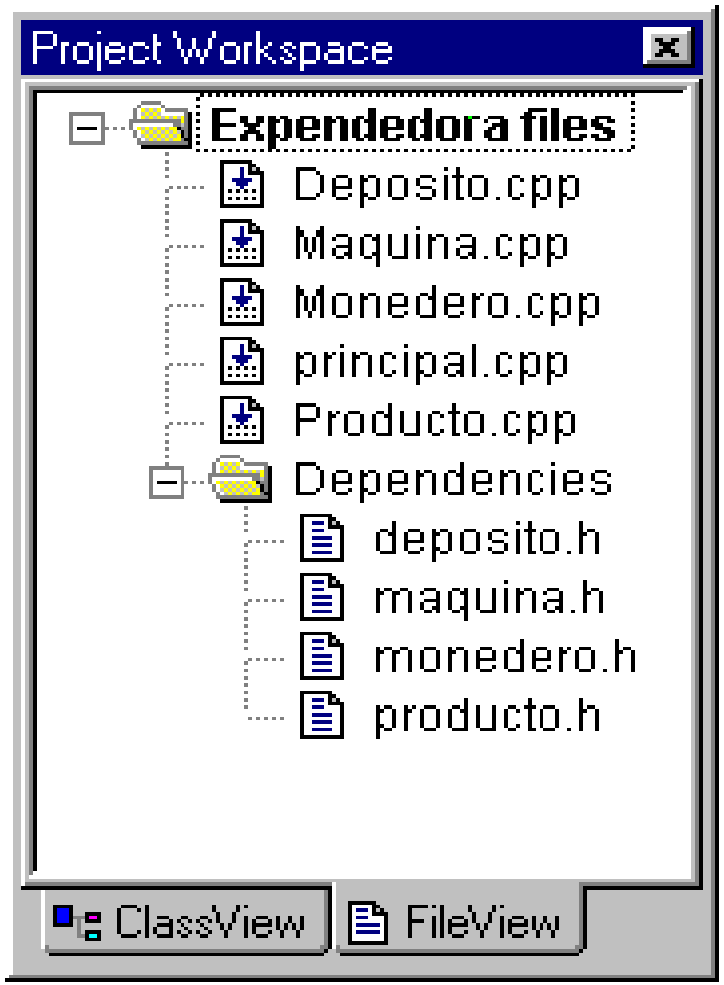
- Genera documentación RTF y HTML de código C++



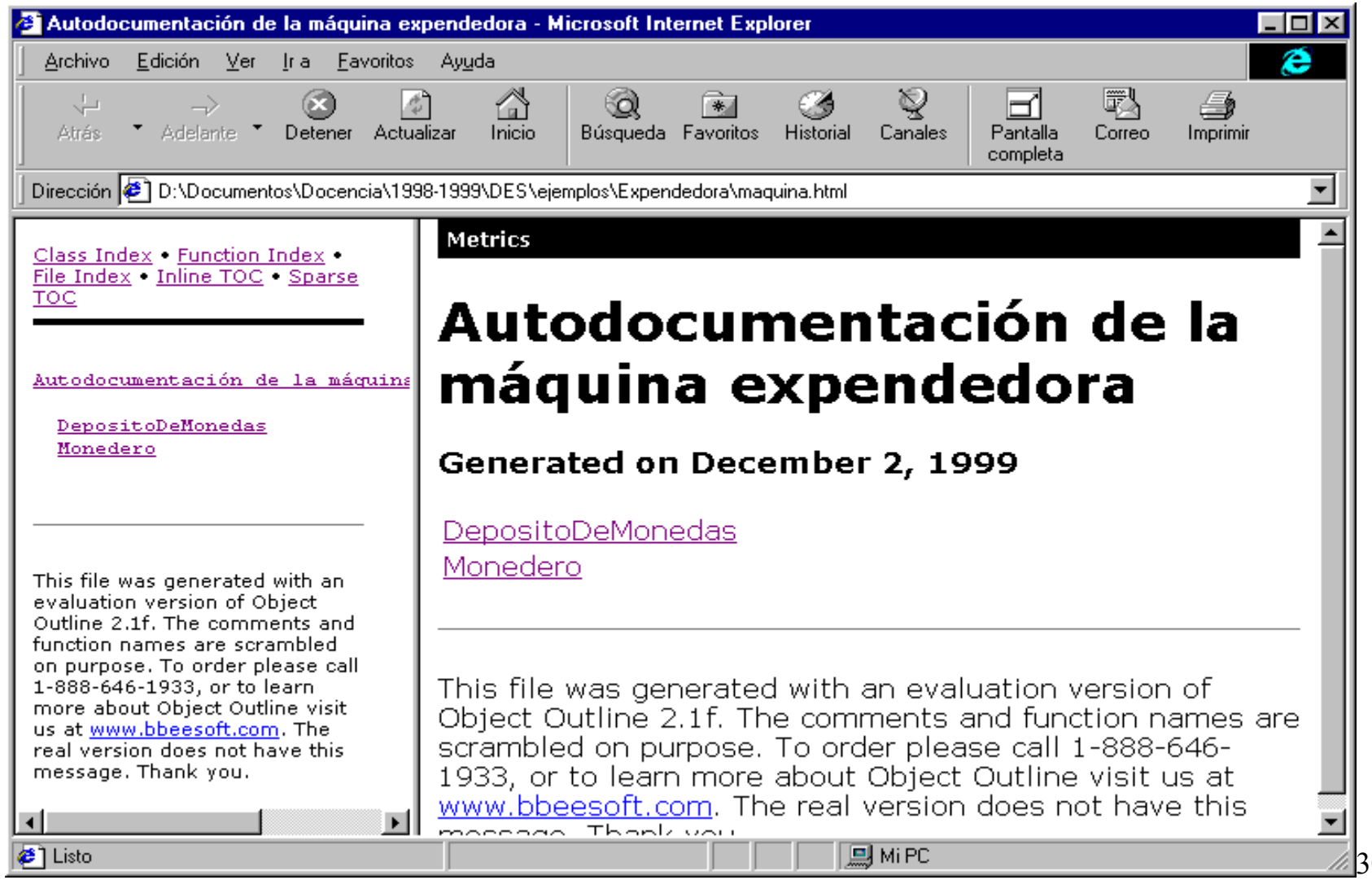
Object outline



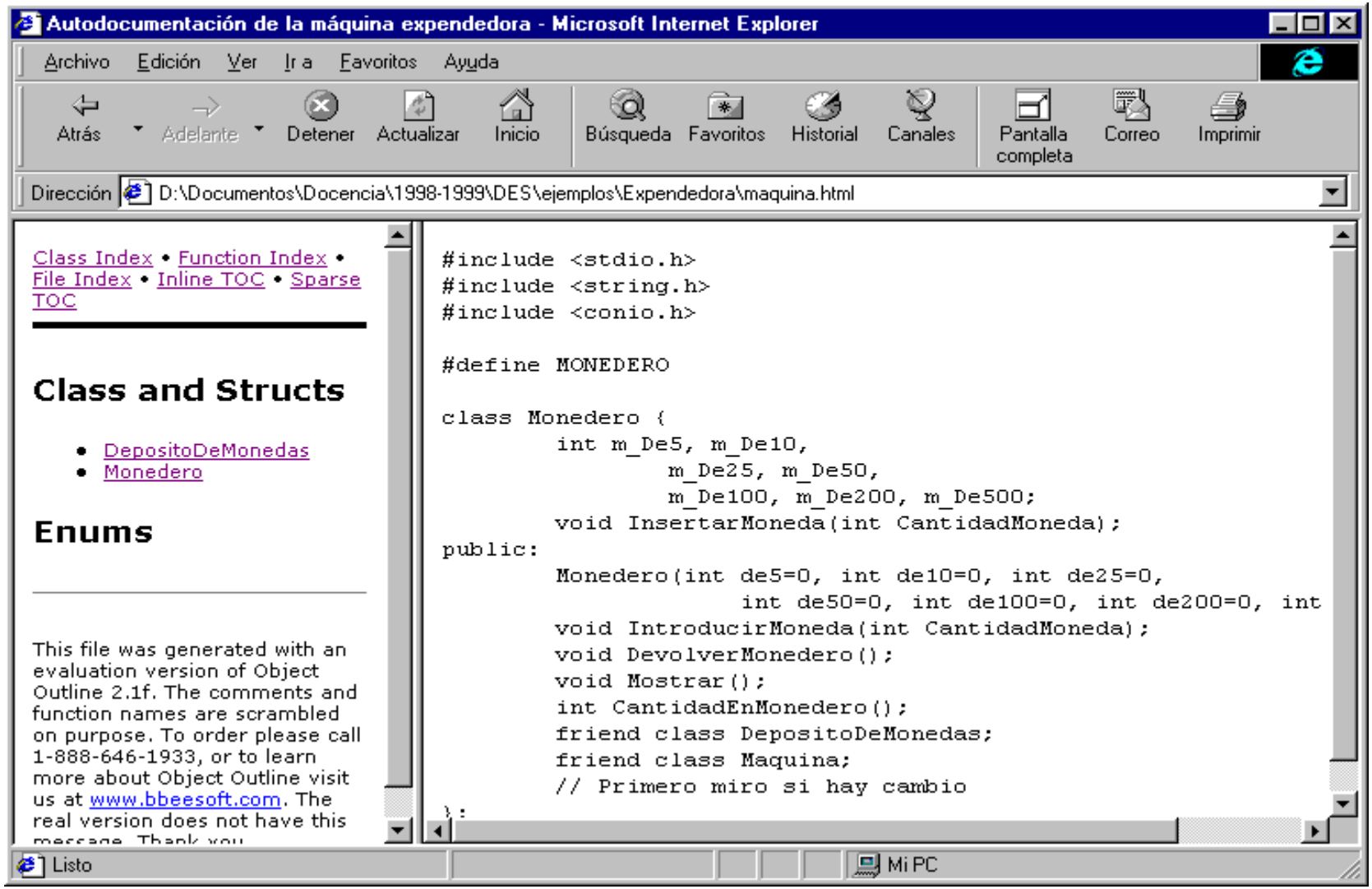
Object outline



Object outline



Object outline



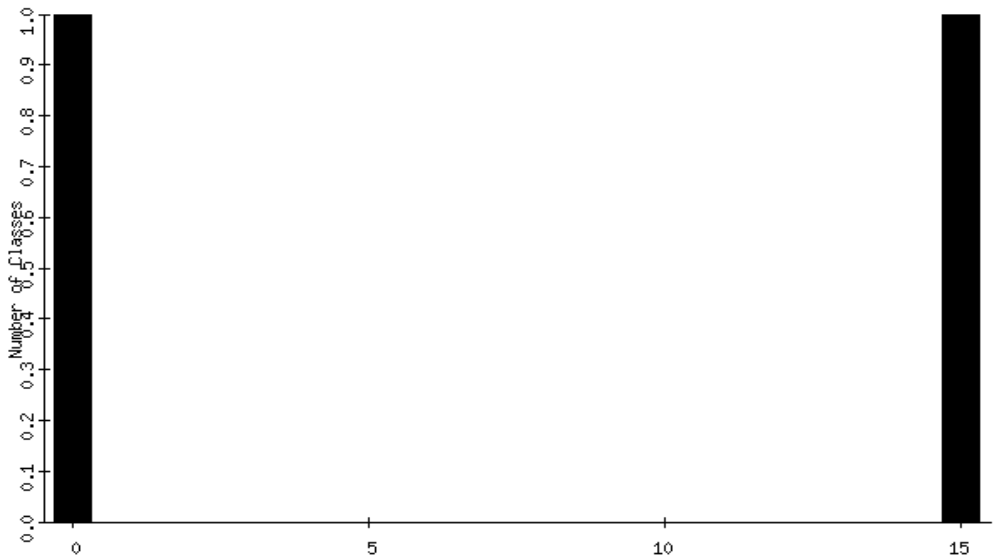
Object outline

Autodocumentación de la máquina expendedora

Metrics For Autodocumentación de la máquina expendedora

- Summary
- Class interface size distribution
- Class size distribution
- Class complexity distribution
- Verbose class metric list with subchapters
- Verbose function metric list with subchapters

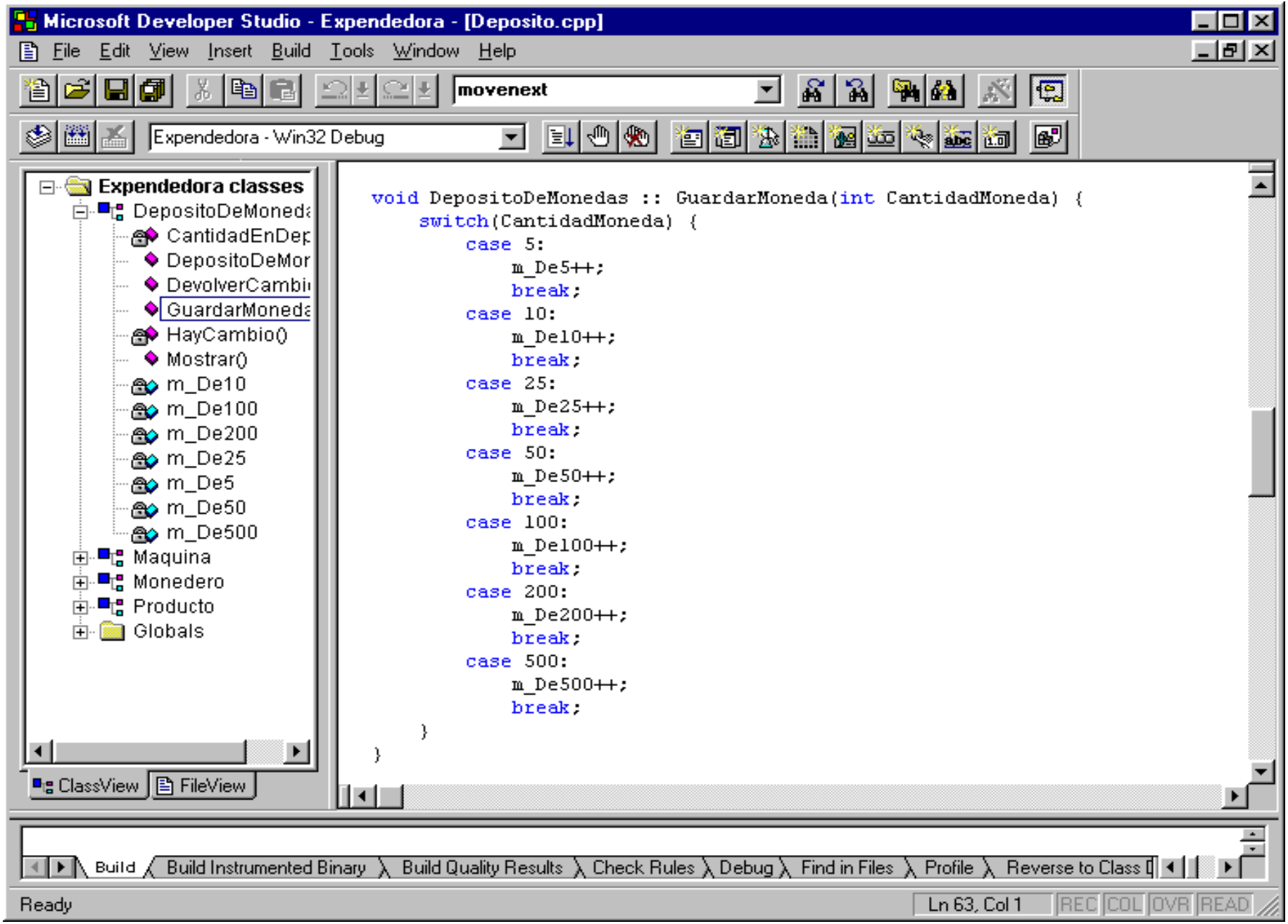
Class McCabe distribution



Object outline

Function Metrics

Class	Statements	McCabe Complexity
DepositoDeMonedas::DevolverCambio	44	2
DepositoDeMonedas::GuardarMoneda	14	8
DepositoDeMonedas::HayCambio	19	2
DepositoDeMonedas::Mostrar	8	1
DepositoDeMonedas::DepositoDeMonedas	7	1
DepositoDeMonedas::CantidadEnDeposito	1	1
Monedero::Monedero	0	0
Monedero::CantidadEnMonedero	0	0
Monedero::DevolverMonedero	0	0
Monedero::IntroducirMoneda	0	0
Monedero::Mostrar	0	0
Monedero::InsertarMoneda	0	0


Microsoft Developer Studio - Expendedora - [Deposito.cpp]
File Edit View Insert Build Tools Window Help
movenext
Expendedora - Win32 Debug
Expendedora classes
DepositoDeMoneda
CantidadEnDep
DepositoDeMor
DevolverCambi
GuardarMoneda
HayCambio()
Mostrar()
m_De10
m_De100
m_De200
m_De25
m_De5
m_De50
m_De500
Maquina
Monedero
Producto
Globals
ClassView
FileView
void DepositoDeMonedas :: GuardarMoneda(int CantidadMoneda) {
switch(CantidadMoneda) {
case 5:
m_De5++;
break;
case 10:
m_De10++;
break;
case 25:
m_De25++;
break;
case 50:
m_De50++;
break;
case 100:
m_De100++;
break;
case 200:
m_De200++;
break;
case 500:
m_De500++;
break;
}
}
Build
Build Instrumented Binary
Build Quality Results
Check Rules
Debug
Find in Files
Profile
Reverse to Class
Ready
Ln 63, Col 1
REC COL OVR READ

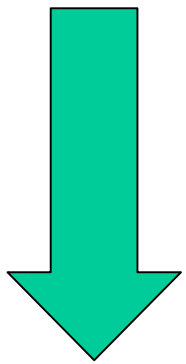
Herramientas para mantenimiento de software

- Generadores de referencias cruzadas.
- Generadores de organigramas.
- Controladores de código fuente.
- Analizadores automáticos de interfaces.
- Gestores de ficheros.
- Descompiladores.
- Evaluadores del impacto de las modificaciones.
- Detectores de componentes afectados por los cambios.

Ingeniería inversa de interfaces de usuario

Ingeniería inversa de interfaces de usuario

Muchos programas gozan de gran fiabilidad, capacidad de procesamiento, etc., pero sus diseñadores han olvidado la comodidad y facilidad de uso del usuario final



"Respetemos la lógica interior, pero adaptemos su aspecto exterior".

Consejos de Plaisant et al.

- Recopilación de la documentación disponible
- Entrevistas a los diferentes usuarios
  - Dirección
  - Diseñadores y equipo de mantenimiento
  - Usuarios finales
- Uso del sistema por el propio equipo de mantenimiento.
- Uso de contadores