Etapas en la vida del software

La necesidad del software suele expresarse inicialmente en lenguaje natural de una manera informal y, por tanto, imprecisa.

A partir de la expresión de la necesidad, la vida del software pasa por *5 etapas*

Análisis y especificación del problema. Se trata de analizar el problema y *concretar la entrada y la salida* del programa de manera precisa. En programas grandes hay *otros* elementos.

Diseño . En la etapa de diseño conviene concretar tres cosas

- Las *estructuras de datos* . En estas primeras etapas usaremos sólo variables como las vistas.
- Los *procedimientos y funciones* (los algoritmos que procesan los datos) expresados en seudocódigo.
- El *lenguaje* a utilizar.

Codificación . En esta etapa, los algoritmos que implementan procedimientos y funciones se deben traducir al lenguaje utilizado usando las estructuras de dados elegidas.

Validación . En realidad, habría que decir 'Validación de la especificación'. Se trata de comprobar si el programa construido responde a todas las entradas que permite la especificación dando una salida correcta. Observar que un programa válido para algunas entradas no tendría por que serlo para otras.

Mantenimiento . Este epígrafe incluye todo el trabajo que se realizará después de que el programa esté funcionando normalmente. Se suelen incluir

- Corrección de los *errores* que se descubren.
- Implementar las *modificaciones* que solicite el cliente
- Implementar las *ampliaciones* que solicite el cliente.

El problema

Escribir el programa que calcula la edad media de los 50 alumnos de una clase.

1 Análisis y especificación

Es decir, tenemos que responder con *precisión* a principalmente a dos preguntas

Qué *datos* toma como *entrada* .

- *cuantos* o *qué* datos son
- de *donde* los toma
  - ya los conoce el programador
  - hay que obtenerlos
    - preguntando al usuario (teclado)
    - leyéndolos de la memoria auxiliar (disco, etc.)
- *cuando* los lee
  - todos al principio,
  - poco a poco a medida que se necesitan etc .

Cuándo y *cómo* se muestran los datos de *salida*

- En pantalla, en impresora. Encolumnados o no etc.
- Se muestran a medida que se obtienen o bien, todos al final etc.

En nuestro ejemplo de la media sería

Entrada:

- Datos: las edades de los alumnos y el número de alumnos
- Cómo se obtienen:
  - las edades preguntando al usuario
  - el número de alumnos es conocido

Salida:

- El mensaje *La edad media es*
- El número que expresa la *edad media* .

2 Diseño

Lenguaje de implementación : Pascal estándar ISO 10206

Estructuras de datos : variables sencillas.

Construcción del algoritmo

Suponemos que hay fichas de alumnos y que en ellas está la edad.
(Recordemos que ya no vamos a decir calculadoras. En adelante, variables).
Aquí podemos usar *variables* para contar los alumnos y sumar las edades.

Para contar los alumnos.

- La vamos a llamar *ctaAlumnos* .
- Valor inicial cero y
- Por cada edad leída sumamos 1.

Para obtener la suma de las edades.

- La vamos a llamar *sumaEdades* .
- Valor inicial cero y
- Sumamos cada edad leída.

Tenemos que poner como valor inicial 0 en ambas variables.

```
sumaEdades <-- 0
ctaAlumnos <-- 0
```

Para cada ficha hay que leer la edad.
Podríamos imaginar que hay una tercera variable a la que llamamos edad en la que ponemos la edad que hay en la ficha. Esta acción se representaría así

```
leer edad
```

Pero esto significa que es un algoritmo/programa de los *interactivos* : se leen datos mientras se ejecuta. Por tanto hay dos personajes: autor y usuario
Si hay un usuario tenemos que mostrarle un *mensaje* que le pide la edad, por tanto la anterior instrucción debe ir precedida de esta otra

```
escribir ' escriba la edad del siguiente alumno '
leer edad
```

El significado de estas dos instrucciones es

El programa/algoritmo, *cuando se está ejecutando* , muestra el mensaje que va entre comillas

El *usuario* del programa, al ver el mensaje, *introduce* el *dato*

El *dato* *se* *almacena* en la variable *edad*

Para cada una de las 50 fichas hay que realizar 4 acciones

```
escribir 'escriba la edad del siguiente alumno'
leer edad
```

```
sumaEdades <-- sumaEdades + edad
ctaAlumnos <-- ctaAlumnos + 1
```

Como sabemos que hay 50 fichas, hay que repetir 50 veces esas instrucciones lo que, teniendo en cuenta que ctaAlumnos tiene el valor inicial 0, se hace así:

```
mientras (ctaAlumnos < 50)
   escribir 'escriba la edad del siguiente alumno'
   leer edad
   sumaEdades <-- sumaEdades + edad
   ctaAlumnos <-- ctaAlumnos + 1
fin mientras
```

Ahora sólo falta mostrar la salida

```
escribir 'La media es ' sumaEdades / ctaAlumnos
```

El significado de esta instrucción es que al ejecutarse

Escribe la expresión que *La media es* (con un espacio al final)

Calcula la media haciendo la división

Muestra la media calculada

Ahora podemos poner todo junto

```
sumaEdades <-- 0
ctaAlumnos <-- 0

mientras (ctaAlumnos < 50)
   escribir 'escriba la edad del siguiente alumno'
   leer edad
   sumaEdades <-- sumaEdades + edad
   ctaAlumnos <-- ctaAlumnos + 1
fin mientras

escribir 'La media es ' sumaEdades / ctaAlumnos
```

Responder a las siguientes preguntas

¿Es correcto el algoritmo?

¿Es capaz este algoritmo de calcular la media de edad de una clase de sesenta alumnos? ¿Y de una de 30 alumnos?

Ejercicio

Modificar en el código anterior para que sea válido para cualquier número de alumnos entre 0 y 1000?

El programa podría preguntar al usuario cuántas fichas hay, mostrando el correspondiente mensaje y almacenar el resultado en una variable que se llame numAlumnos.

```
escribir ('¿cuantos alumnos hay?')
leer numAlumnos
```

La comparación debería consistir en enfrentar el número de alumnos ya procesados con el total de alumnos almacenado en numAlumnos

```
mientras (ctaAlumnos < numAlumnos)
```

El algoritmo modificado sería

```
sumaEdades <-- 0
ctaAlumnos <-- 0

 escribir '¿cuantos alumnos hay?'
leer numAlumnos

mientras ( ctaAlumnos < numAlumnos )
   escribir 'escriba la edad del siguiente alumno'
   leer edad
   sumaEdades <-- sumaEdades + edad
   ctaAlumnos <-- ctaAlumnos + 1
fin mientras

escribir 'La media es ' sumaEdades / ctaAlumnos
```

¿Es VÁLIDO este algoritmo para la especificación pedida?
(Recordar que la ESPECIFICACIÓN exige que sea válido para cualquier número de alumnos entre 0 y 1000)

¿Qué ocurre si la *clase tiene cero alumnos* ?

Si hay 0 alumnos en ctaAlumnos habrá 0. Como no se puede dividir por 0 hay que separar los *dos casos* posibles en la especificación que exigen tratamiento distinto:

cero alumnos

1 a 1000 alumnos

La salida debería ser así

```
si (ctaAlumnos = 0)
    escribir 'No hay media: la clase tiene 0 alumnos'
si no
    escribir 'La media es ' sumaEdades/ctaAlumnos
fin si
```

Por tanto, el algoritmo VALIDADO PARA LA ESPECIFICACIÓN sería

```
sumaEdades <-- 0
ctaAlumnos <-- 0

 escribir '¿cuantos alumnos hay?'
leer numAlumnos

mientras ( ctaAlumnos < numAlumnos )
   escribir 'escriba la edad del siguiente alumno'
   leer edad
   sumaEdades <-- sumaEdades + edad
   ctaAlumnos <-- ctaAlumnos + 1
fin mientras

si (ctaAlumnos = 0)
    escribir 'No hay media: la clase tiene 0 alumnos'
si no
    escribir 'La media es ' sumaEdades/ctaAlumnos
fin si
```

¿Por qué queremos tratar el caso RARO de que una clase tenga 0 alumnos?

Supongamos que el programa se ha ampliado para calcular las medias de todas las clases de una Facultad el nombre de la clase (1A, 1B, 1C, 2A, 2B, 3A, 3B) está codificado en el algoritmo de modo que sucesivamente el algoritmo

- muestra el mensaje *'Ahora vamos a trabajar con la clase ' clase*
- clase es una *calculadora alfanumérica* que permite almacenar en su memoria de pantalla valores como 1A, 3B...
- después pide los datos de la clase, hace el proceso y muestra su media

Con una clase vacía,

- el *primer diseño* provocaría un *error* deteniendo el proceso al llegar a la clase vacía y, por tanto, no procesando las clases posteriores.
- el *segundo diseño* procesaría todas las clases correctamente.