5. Diseño Detallado

Introducción

El diseño detallado es una fase del proceso de desarrollo de software que sigue al diseño de alto nivel, y que se centra en especificar la lógica y los algoritmos detallados que serán necesarios para implementar el software deseado. Se encarga de aspectos más específicos de cómo funcionará el SW, como la lógica interna de cada componente.

Diseño Detallado y PDL

El PDL o Process Design Language es un lenguaje preciso, simple, **independiente a la implementación pero descriptivo del mismo**. Sirve para especificar la lógica de un componente con mucha precisión, pero sin entrar en detalles de implementación específicos del lenguaje de programación.

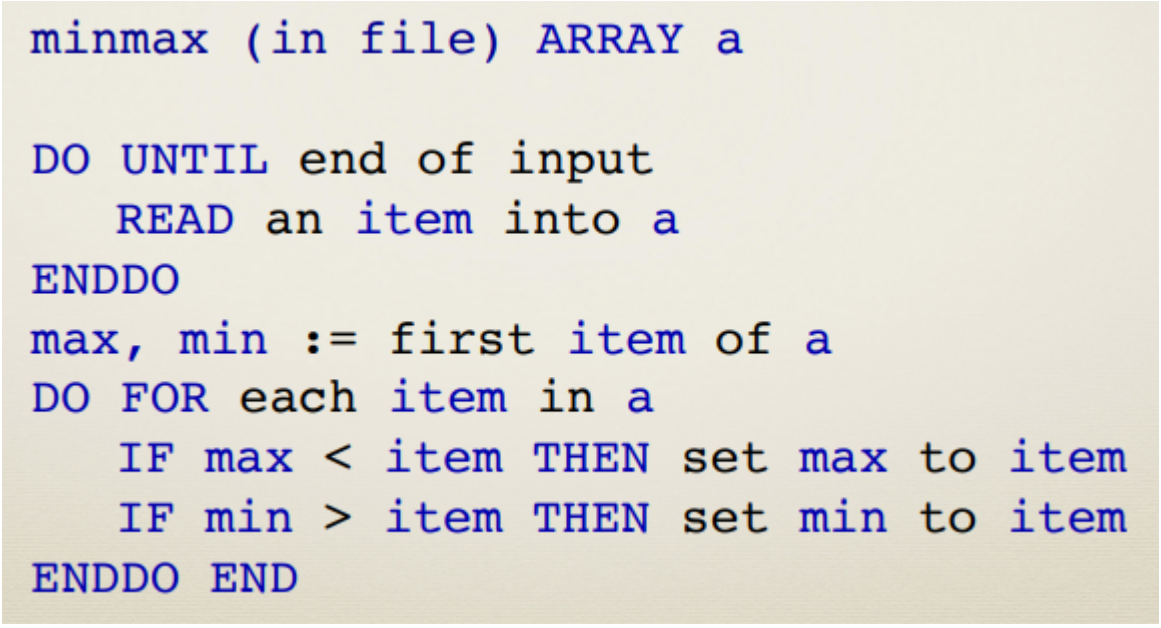
Este lenguaje tiene la sintaxis de un lenguaje de programación funcional, pero es un lenguaje natural, parecido a un inglés estructurado.

Sirve para:

- Especificar diseño en un grado específico.
- Es un medio de comunicación entre diseñadores, desarrolladores, etc.
- Comprende un nivel de abstracción intermedio, donde se define la lógica de un componente con suficiente detalle como para comprenderla, pero sin entrar en detalles específicos de codificación.
- Es compatible con la técnica top-down.
- Es automatizable en algunas de sus partes ya que PDL puede ser procesado automáticamente por herramientas de SW, lo que facilita la verificación y generación de código.
- Independiente del lenguaje de programación, ya que PDL se centra en la lógica de un componente.
- Facilita la captura de la lógica completa de un procedimiento ya que se pueden representar condiciones, bucles, estructuras de control y flujos de datos de manera estructurada.

- Permite adaptarse a enfoques de desarrollo incremental y modular. Facilita la descripción de módulos y procesos independientes.

Ejemplo de lógica en PDL: determinar el mínimo y máximo de un conjunto de números en un archivo.



Constructores de PDL

- IF-THEN-ELSE: describir toma de decisiones de un proceso.

```
IF condición THEN
    Realizar acción A
ELSE
    Realizar acción B
ENDIF
```

  Las condiciones y sentencias no necesitan escribirse formalmente.

- DO: indicar repeticiones o bucles.

```
DO WHILE "there are chars in the input"
    Realizar una o más acciones
ENDDO

DO UNTIL "the end of file is reached"
    Realizar una o más acciones
ENDDO

DO FOR "each item in the list EXCEPT when item is zero"
```

```
    Realizar una o más acciones
ENDDO
```

El criterio de iteración no necesita establecerse formalmente.

- CASE: manejar múltiples opciones o casos generales en una estructura de selección

```
CASE OF variable
    WHEN valor1
        Realizar acción A
    WHEN valor2
        Realizar acción B
    OTHERWISE
        Realizar acción predeterminada
ENDCASE
```

💡 Todos los constructores deben ser programables.

Proceso de detallado de componentes

El desarrollo se realiza gradualmente y es un **refinamiento paso a paso**. Se comienza por convertir la especificación del módulo en una descripción abstracta del algoritmo.

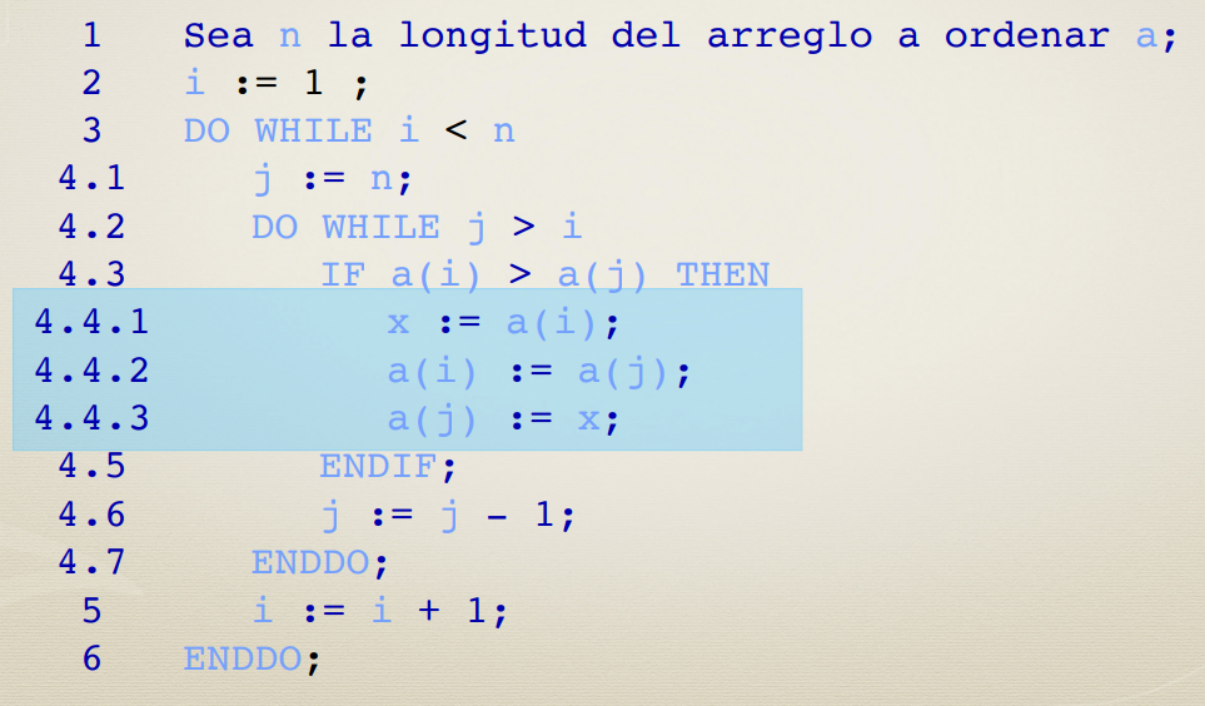
En cada paso, se descompone una o más sentencias del algoritmo actual en instrucciones más detalladas.

Luego, se termina cuando todas las instrucciones son lo suficientemente precisas como para llevarlas fácilmente al lenguaje de programación.

Ejemplo:

```
1   Sea n la longitud del arreglo a ordenar a;
2   i := 1 ;
3   DO WHILE i < n
4      encontrar el menor de ai...an,
          e intercambiarlo con el
          elemento de la posición i;
5      i := i + 1;
6   ENDDO;
```

```
1   Sea n la longitud del arreglo a ordenar a;
2   i := 1 ;
3   DO WHILE i < n
4.1    j := n;
4.2    DO WHILE j > i
4.3       IF a(i) > a(j) THEN
4.4          intercambiar los elementos
                en las posiciones j e i;
4.5       ENDIF;
4.6       j := j - 1;
4.7    ENDDO;
 5     i := i + 1;
 6  ENDDO;
```



Verificación del Diseño

Un diseño es correcto cuando cumple las especificaciones del sistema. Hay **tres métodos de verificación**

1. Recorrido del diseño: reunión informal entre diseñador y líder, recorriendo todo el diseño
2. Revisión crítica del diseño: sólo si esta escrito en PDL, se revisa consistencia con especificación a niveles específicos (existencia de interfaces, módulos, etc)
3. Verificación de consistencia: sigue un proceso de revisión estándar. Se usan listas de control.

Métricas

1. Complejidad ciclomática

   Mide la complejidad de un módulo, depende de las condiciones y sentences de control. A medida que estas aumentan, la complejidad aumenta.

2. Vínculos de datos

   Este tipo de métricas capturan la interacción de datos entre las distintas porciones del SW.

3. Métrica de cohesión

Mide la dependencia intramodular de elementos. El valor será mas alto si cada ejecución posible del módulo usa todos los recursos (variables) del módulo.