1.- Diseño del software

El *diseño de software* es el primer paso de la **fase de desarrollo** de cualquier producto o sistema de ingeniería.
Una vez que se han establecido los requerimientos del software, el diseño del software es la primera de tres actividades técnicas: *diseño*, *codificación* y *prueba*. Los requerimientos del programa, establecidos mediante los modelos de información, funcional y de comportamiento, alimentan el paso de diseño.
El *diseño de datos* transforma el modelo del campo de información, creado durante el análisis, en las estructuras de datos que se van a requerir para implementar el software. El *diseño arquitectónico* define las relaciones entre los principales elementos estructurales del programa. El *diseño procedimental* transforma los elementos estructurales en una descripción procedimental del software. Se genera el código fuente y, para integrar y validar el software, se llevan a cabo las pruebas.

El proceso de diseño

Diseñar un sistema es determinar un conjunto de componentes y de interfaces entre componentes que satisfagan un conjunto específico de requerimientos.
El *diseño del software* se realiza en dos pasos: El *diseño preliminar* se centra en la transformación de los requisitos en los datos y en la arquitectura del software. El *diseño detallado* se ocupa del refinamiento de la representación arquitectónica que lleva a una estructura de datos detallada y a las representaciones algorítmicas del software.
Además del *diseño de datos*, del *diseño arquitectónico*, y del *diseño procedimental*, muchas aplicaciones requieren una actividad de *diseño de interfaz.* El *diseño de interfaz* establece la disposición y los mecanismos para la interacción hombre-máquina.

Metodologías para el diseño:

Diseño estructurado:

El objetivo del diseño estructurado es proporcionar un enfoque sistemático para la obtención de las estructuras de programa. Una visión global del software y la base del diseño preliminar.
Permite una cómoda transacción de las representaciones de la información (por ejemplo, el diagrama de flujo de datos) contenida en una *especificación de requisitos del software* a una descripción de diseño de la estructura del programa

Herramientas:

- Modelo de Cartas de estructura
- Modelo relacional
- Lenguaje de diseño de programas (PDL)
- Modelo de interfaz

Diseño orientado a objetos:

A diferencia de los métodos de diseño de software convencionales, el DOO proporciona un diseño que alcanza diferentes niveles de modularidad. La mayoría de los componentes de un sistema, están organizados en subsistemas, un «módulo» a nivel del sistema. Los datos y las operaciones que manipulan los datos se encapsulan en objetos. Además, el DOO debe describir la organización específica de los datos de los atributos y el detalle procedural de cada operación.
Tareas del DOO:

- Identificar y definir clases y objetos (atributos y comportamiento).
- Identificar las interacciones y relaciones entre objetos y clases, sus asociaciones, composiciones, agregaciones y relaciones de herencia.
- Considerar requerimientos no funcionales (performance, seguridad, restricciones de estrada y salida, etc.).
- Insertar bibliotecas que la clase detalla.
- Considerar las componentes reutilizables (previamente construidas y a construir).
- Considerar los requerimientos de la interfaz de usuario.
- Considerar la gestión de tareas (dirigida por evento o por tiempo).

Herramientas:

- Diagrama de clases
- Diagrama de objetos
- Diagrama de paquetes
- Diagrama de interacción
- Diagrama de secuencia
- Diagrama de colaboración
- Diagrama de actividades
- Lenguaje de diseño de programas (PDL)

2.- Codificación

2.1.- El proceso de traducción

El paso de codificación traduce una representación del software, dada por un diseño detallado, a una realización en un lenguaje de programación. El proceso de traducción continúa cuando un compilador acepta el *código fuente* como entrada y produce como salida un *código objeto* dependiendo de la máquina.

Elección de un lenguaje

Entre los criterios que se aplican durante la evaluación de los lenguajes disponibles están:

1. Área de aplicación general
2. Complejidad algorítmica y computacional
3. Entorno en el que se ejecutará el software
4. Consideraciones del rendimiento
5. Complejidad de las estructuras de datos
6. Conocimiento de la plantilla de desarrollo de software
7. Disponibilidad de un buen compilador o compilador cruzado.

El área de aplicación de un proyecto es el criterio que más se aplica durante el proceso de selección del lenguaje.

Lenguajes de programación e ingeniería del software

Independientemente del paradigma de ingeniería del software, el lenguaje de programación tendrá impacto en la planificación, el análisis, el diseño, la codificación, la prueba y el mantenimiento de un proyecto. El papel del lenguaje de programación ha de tenerse presente en todo momento. Sin embargo, la calidad del resultado final se encuentra más fuertemente unida a las actividades de ingeniería del software que preceden y siguen a la codificación.

Durante el paso de planificación del proyecto, raramente se toman en consideración las características técnicas de un lenguaje de programación. Sin embargo, la planificación de las herramientas de soporte asociadas con la definición de recursos puede requerir que se especifique un compilador en particular (y su software) o un entorno de programación. La estimación de costos y de la agenda puede requerir que se ajuste la curva de aprendizaje debido a la inexperiencia de la plantilla con un determinado lenguaje.

La calidad de un diseño de software viene dada por su independencia de las características de los lenguajes de programación. (Una excepción notable es el diseño orientado a los objetos). Sin embargo, los atributos del lenguaje juegan,

de hecho, un papel en la calidad de un diseño acabado y afectan (tanto consciente como inconscientemente) a la forma de especificar el diseño.
Las características de un lenguaje de programación en los pasos que componen la prueba del software es difícil de prever. Los lenguajes que soportan directamente las construcciones estructuradas tienden a reducir la complejidad ciclomática de un programa, haciéndolo, de alguna forma, más fácil de probar. Los lenguajes que soportan la especificación de subprogramas y procedimientos externos hacen que la prueba de integración sea mucho menos propensa a errores.
Al igual que ocurre con la prueba, todavía no se comprende totalmente el efecto de las características de los lenguajes de programación sobre el mantenimiento del software. Sin embargo, no hay duda de que las características técnicas pueden mejorar la legibilidad del código y reducir la complejidad, lo que es importante para un mantenimiento efectivo.

2.2.- El estilo de codificación

La función de un módulo debe resultar clara sin necesidad de referirse a ninguna especificación del diseño. En otras palabras, el código debe ser comprensible. El estilo de codificación conlleva una filosofía de codificación que mezcle la simplicidad con la claridad.

Documentación del código

La documentación del código fuente comienza con la elección de los nombres de los identificadores (variables y etiquetas), continúa con la localización de los comentarios y termina con la organización visual del programa (indentación).

Declaración de datos

La complejidad y la organización de las estructuras de datos se definen durante el paso de diseño. El estilo en la declaración de datos se establece cuando se genera el código.
El orden de las declaraciones de datos se debe estandarizar incluso aunque el lenguaje de programación no tenga requisitos específicos.
El orden hace que los atributos sean fáciles de descubrir, comprobar, depurar y mantener.
Cuando se declaran múltiples nombres de variables en una sola sentencia, merece la pena ponerlos en orden alfabético.

Construcción de sentencias

La construcción del flujo lógico del software se establece durante el diseño. La construcción de sentencias individuales, sin embargo, es parte del paso de codificación. La construcción de sentencias se debe basar en una regla general: cada sentencia debe ser simple y directa.

Entrada / salida

El estilo de la entrada y la salida se establece durante el análisis de requisitos del software y el diseño. Sin embargo, la forma en que se implementa la E/S puede ser una característica determinada de la aceptación del sistema por una comunidad de usuarios. El estilo de la entrada y la salida variará con el grado de interacción humana.