REPÚBLICA DEL ECUADOR

Ministerio de Educación

UNIDAD EDUCATIVA "BAÑOS"

Baños de Agua Santa – Tungurahua

PROYECTO INTERMODULAR

"Temas del Currículo del Módulo Formativo Nº3: Programación y Base de Datos"

Figura Profesional: Informática

Elaborado por: Anahi Sailema

Pamela Aponte

Elkyn Zambrano

Curso: Segundo "H"

Módulo formativo:

Módulo 3: Programación y Base de Datos

Docentes Técnicos: Mg. Diana Fuentes

Año Lectivo: 2024-2025

EL NUEVO ECUADOR

Índice

Programación

- ✓ **Datos:** En programación, los datos son valores o información que se utilizan para realizar cálculos, tomar decisiones o almacenar información. Los datos pueden ser de diferentes tipos, como números enteros, números decimales, texto, fechas, horas, etc.

Tipos de datos comunes:

1. Números enteros (int): 1, 2, 3, etc.

2. Números decimales (float): 3.14, -0.5, etc.

3. Texto (string): "hola", "adiós", etc.

4. Booleanos (bool): verdadero (true), falso (false)

5. Fechas y horas (date, datetime): 2022-07-25, 14:30:00, etc.

Ejemplo:

Supongamos que queremos crear un programa que calcule el área de un rectángulo. Para hacer esto, necesitamos almacenar los datos de la longitud y el ancho del rectángulo.

```
// Declaración de variables
longitud = 5; // número entero
ancho = 3.5; // número decimal
// Cálculo del área
área = longitud * ancho;
// Impresión del resultado
print("El área del rectángulo es:", área);
```

En este ejemplo, los datos son:

- longitud: un número entero (5)

- ancho: un número decimal (3.5)

- área: el resultado del cálculo, que también es un número decimal

✓ **Algoritmo:** Un algoritmo es un conjunto de instrucciones paso a paso que se utilizan para resolver un problema o realizar una tarea específica en programación. Un algoritmo define la lógica y la secuencia de operaciones que se deben realizar para obtener un resultado deseado.

Características de un algoritmo:

1. Secuencia de pasos: Un algoritmo se compone de una serie de pasos que se deben seguir en un orden específico.

2. Entradas y salidas: Un algoritmo recibe entradas (datos) y produce salidas (resultados).

3. Lógica y toma de decisiones: Un algoritmo utiliza lógica y toma de decisiones para procesar las entradas y producir las salidas.

Ejemplo:

Supongamos que queremos crear un algoritmo para calcular el mayor de dos números.

// Algoritmo para calcular el mayor de dos números

1. Recibir dos números como entradas (a y b)

2. Comparar a y b

3. Si a es mayor que b, entonces:

- Salida: a es el mayor

4. Si b es mayor que a, entonces:

- Salida: b es el mayor

5. Si a y b son iguales, entonces:

- Salida: a y b son iguales

En este ejemplo, el algoritmo:

- Recibe dos números como entradas (a y b)
- Compara a y b utilizando lógica y toma de decisiones
- Produce una salida que indica cuál es el mayor de los dos números

- ✓ **Pseudocódigo:** Un pseudocódigo es una representación simbólica y simplificada de un algoritmo o un programa de computadora. Se utiliza para describir la lógica y la secuencia de operaciones de un programa de manera clara y concisa, sin utilizar un lenguaje de programación específico.

Características de los pseudocódigos:

1. Simbólico: Los pseudocódigos utilizan símbolos y abreviaturas para representar las operaciones y los datos.

2. Simplificado: Los pseudocódigos omiten detalles innecesarios y se enfocan en la lógica y la secuencia de operaciones.

3. Independiente del lenguaje: Los pseudocódigos no están vinculados a un lenguaje de programación específico.

4. Fácil de entender: Los pseudocódigos están diseñados para ser fáciles de entender y analizar.

Ejemplo de pseudocódigo:

Supongamos que queremos crear un pseudocódigo para calcular el área de un rectángulo:

INICIO

LEER longitud

LEER ancho

CALCULAR área = longitud * ancho

IMPRIMIR área

FIN

En este ejemplo, el pseudocódigo:

- Utiliza símbolos y abreviaturas para representar las operaciones y los datos.
- Omite detalles innecesarios y se enfoca en la lógica y la secuencia de operaciones.
- No está vinculado a un lenguaje de programación específico.
- Es fácil de entender y analizar.

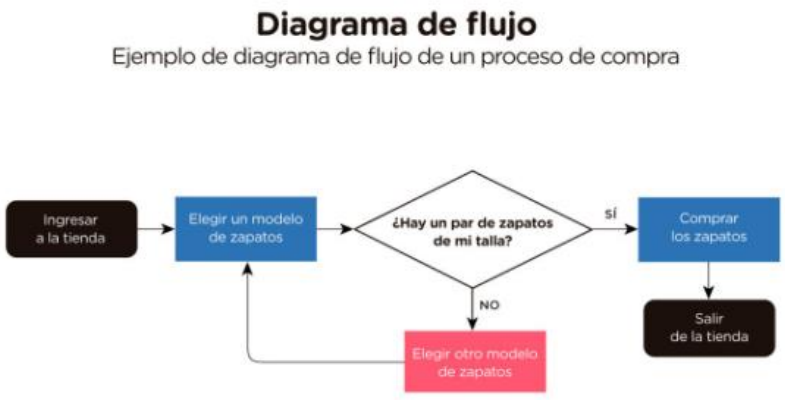
✓ **Flujograma:** Un flujograma o diagrama de flujo es una herramienta gráfica que representa visualmente un proceso o un algoritmo. Es muy utilizado en los campos de la informática, la economía, la industria e incluso la psicología, para organizar de un modo simple las decisiones y los pasos involucrados en algún tipo de proceso.

Ejemplo:

Ejemplo de flujograma de un proceso de compra:

Figura 1

Flujograma



Nota: Se puede observar un flujograma de un proceso de compras. Fuente: (Concepto,2025).

- ✓ **Estructuras de control:** Las estructuras de control son un conjunto de reglas que hacen posible la gestión del flujo de ejecución de los programas, esto lo logran determinando el orden en que las instrucciones se ejecutan. *Grosso modo*, las estructuras de control permiten que el código tome decisiones, repita acciones o bien siga su flujo de ejecución secuencial.

  Ejemplo:

  Ejemplo de estructura de control secuencial:

  Las estructuras de control secuenciales son las más básicas. En este caso, las instrucciones se ejecutan una tras otra en el mismo orden en que están escritas. Piensa en este tipo de estructuras como una fila de acciones que siguen un patrón fijo. Estas son la base de cualquier algoritmo, en donde cada acción sigue a la anterior de modo predecible. Veamos un ejemplo en JavaScript:

  ```
  // Ejemplo en JavaScript
  let nombre = "María";
  let saludo = "Hola, " + nombre + "!";
  console.log(saludo);
  ```

  En este ejemplo, la primera línea asigna un valor a la variable nombre, luego se crea un saludo y finalmente se imprime en la consola. Todas las acciones ocurren en el orden en que fueron escritas.

- ✓ **Operadores:** En este ejemplo, la primera línea asigna un valor a la variable nombre, luego se crea un saludo y finalmente se imprime en la consola. Todas las acciones ocurren en el orden en que fueron escritas.

  Ejemplo:

  Operadores aritméticos que se utilizan para realizar operaciones matemáticas en programación.

- **+ (suma):** Se utiliza para sumar dos valores.
- **- (resta):** Se utiliza para restar el valor de la derecha del valor de la izquierda.
- *** (multiplicación):** Se utiliza para multiplicar dos valores.
- **/ (división):** Se utiliza para dividir el valor de la izquierda por el valor de la derecha.
- **% (módulo):** Devuelve el resto de la división del valor de la izquierda por el valor de la derecha.
- **// (división entera):** Devuelve el cociente entero de la división del valor de la izquierda por el valor de la derecha.
- **** (exponenciación):** Se utiliza para elevar el valor de la izquierda a la potencia del valor de la derecha.

Figura 2

Operadores aritméticos

Nota: Ejemplo operador aritmético «suma +». Fuente: (Apinem,2025).

✓ **Funciones:** En programación, una función es un conjunto de instrucciones que realiza una tarea específica. Puedes imaginar una función como una pequeña máquina que toma ciertos datos (argumentos), realiza una operación y devuelve un resultado. Utilizar funciones nos permite escribir código más limpio, organizado y fácil de entender.

Ejemplo:

```
# Ejemplo de una función en Python
def saludar(nombre):
    return "¡Hola, " + nombre + "!"
# Llamada a la función
mensaje = saludar("Ichibytes")
print(mensaje)
```

En este ejemplo, la función saludar toma un argumento (nombre) y devuelve un saludo personalizado.

- ✓ **Estructura modular de programas:** La estructura modular de programas es una técnica de diseño de software que consiste en dividir un programa en módulos o unidades independientes, cada uno con una función específica. Cada módulo es un conjunto de instrucciones que realizan una tarea particular y pueden ser utilizados por otros módulos o por el programa principal.

  Características de la estructura modular

  1. Independencia: Cada módulo es independiente y puede ser modificado o reemplazado sin afectar a otros módulos.

  2. Reutilización: Los módulos pueden ser reutilizados en otros programas o proyectos.

  3. Flexibilidad: La estructura modular permite agregar o eliminar módulos según sea necesario.

  4. Mantenimiento: La estructura modular facilita el mantenimiento y la depuración de los programas.

  Ejemplo de estructura modular

  Supongamos que queremos crear un programa que realice las siguientes tareas:

  1. Leer un archivo de texto

  2. Procesar el contenido del archivo

  3. Guardar el resultado en un nuevo archivo

  Podemos dividir el programa en tres módulos independientes:

  Módulo 1: Leer archivo

  ```
  def leer_archivo(nombre_archivo):
  # Código para leer el archivo
   return contenido
  ```

Módulo 2: Procesar contenido

```
def procesar_contenido(contenido):

# Código para procesar el contenido

return resultado
```

Módulo 3: Guardar resultado

```
def guardar_resultado(resultado, nombre_archivo):

# Código para guardar el resultado en un nuevo archivo
```

Programa principal

```
def main():

nombre_archivo = "archivo.txt"

contenido = leer_archivo(nombre_archivo)

resultado = procesar_contenido(contenido)

guardar_resultado(resultado, "resultado.txt")

main()
```

En este ejemplo, cada módulo es independiente y realiza una tarea específica. El programa principal utiliza los módulos para realizar la tarea completa.

Lenguaje de programación

- ✓ **Entidades que maneja el lenguaje:** Las entidades que maneja un lenguaje de programación son los elementos básicos que se utilizan para crear programas. Estas entidades pueden ser:

  - Variables: Espacios de memoria que almacenan valores.

  - Constantes: Valores que no cambian durante la ejecución del programa.

  - Tipos de datos: Categorías que definen el tipo de valor que puede almacenar una variable.

  - Operadores: Símbolos que se utilizan para realizar operaciones aritméticas, lógicas, etc.

  - Estructuras de control: Instrucciones que controlan el flujo de ejecución del programa.

  Ejemplo:

```
// Variables
int x = 5;
string nombre = "Juan";
// Constantes
const int PI = 3.14;
// Tipos de datos
int[] edades = {25, 30, 35};
// Operadores
int suma = 2 + 3;
// Estructuras de control
if (x > 10) {
Console.WriteLine("x es mayor que 10");
}
```

- ✓ **Tipos de variables:** Los tipos de variables son las categorías que definen el tipo de valor que puede almacenar una variable. Algunos ejemplos de tipos de variables son:

- Enteros (int): Números enteros, como 1, 2, 3, etc.

- Números decimales (float): Números decimales, como 3.14, -0.5, etc.

- Cadenas de texto (string): Secuencias de caracteres, como "hola", "adiós", etc.

- Booleanos (bool): Valores lógicos, como verdadero (true) o falso (false).

Ejemplo:

```
int edad = 25; // variable entera
float altura = 1.75; // variable decimal
string nombre = "Juan"; // variable cadena de texto
bool isAdmin = true; // variable booleana
```

- ✓ **Estructuras de datos:** Las estructuras de datos son formas de organizar y almacenar datos en un programa. Algunos ejemplos de estructuras de datos son:

- Arreglos (arrays): Colecciones de valores del mismo tipo que se almacenan en una secuencia.

- Listas (lists): Colecciones de valores del mismo tipo que se almacenan en una secuencia y pueden ser modificadas.

- Diccionarios (dictionaries): Colecciones de pares clave-valor que se almacenan en una tabla.

Ejemplo:

```
int[] edades = {25, 30, 35}; // arreglo de enteros
string[] nombres = {"Juan", "María", "Pedro"}; // arreglo de cadenas de texto
Dictionary<string, int> edadPorNombre = new Dictionary<string, int>()
{
   {"Juan", 25},
   {"María", 30},
   {"Pedro", 35}
}; // diccionario de cadenas de texto y enteros
```

- ✓ **Instrucciones del lenguaje:** Las instrucciones del lenguaje son las órdenes que se utilizan para controlar el flujo de ejecución del programa. Algunos ejemplos de instrucciones del lenguaje son:

- Instrucciones de asignación: Asignan un valor a una variable.

- Instrucciones de control de flujo: Controlan el orden en que se ejecutan las instrucciones.

- Instrucciones de iteración: Repiten un conjunto de instrucciones varias veces.

Ejemplo:

```
int x = 5; // instrucción de asignación
if (x > 10) { // instrucción de control de flujo
   Console.WriteLine("x es mayor que 10");
} else {
   Console.WriteLine("x es menor o igual que 10");
}
for (int i = 0; i < 5; i++) { // instrucción de iteración
   Console.WriteLine("Iteración " + i);
}
```

- ✓ **Función y sintaxis:** Una función es un bloque de código que se puede llamar varias veces desde diferentes partes del programa. La sintaxis de una función varía según el lenguaje de programación.

  Ejemplo en C#:

```
// Declaración de la función
public static void Saludar(string nombre)
{
   Console.WriteLine("Hola, " + nombre);
}

// Llamada a la función
Saludar("Juan");
```

  En este ejemplo, se declara una función llamada Saludar que recibe un parámetro de tipo string llamado nombre. La función utiliza el método Console.WriteLine para imprimir un mensaje de saludo en la consola. Luego, se llama a la función pasando el argumento "Juan".

- ✓ **Funciones y librerías básicas:** Las funciones y librerías básicas son bloques de código predefinidos que se pueden utilizar para realizar tareas comunes.

  Ejemplo en C#:

```
// Uso de la función Math.Sqrt
```

```
double raizCuadrada = Math.Sqrt(16);
Console.WriteLine("La raíz cuadrada de 16 es " + raizCuadrada);

// Uso de la librería System.IO
using System.IO;
File.WriteAllText("archivo.txt", "Hola, mundo!");
```

En este ejemplo, se utiliza la función Math.Sqrt para calcular la raíz cuadrada de 16. Luego, se utiliza la librería System.IO para crear un archivo llamado archivo.txt y escribir el texto “Hola, ¡mundo!" en él.

- ✓ **Documentación de programas:** La documentación de programas es el conjunto de comentarios y descripciones que se agregan al código para explicar su funcionamiento y propósito.

  Ejemplo en C#:

```
/// <summary>
/// Esta función saluda a una persona.
/// </summary>
/// <param name="nombre">El nombre de la persona.</param>
public static void Saludar(string nombre)
{
   Console.WriteLine("Hola, " + nombre);
}
```

  En este ejemplo, se agrega una documentación a la función Saludar utilizando comentarios XML. La documentación describe el propósito de la función y el parámetro que recibe.

- ✓ **Descripción de estructuras de datos utilizadas:** Las estructuras de datos son formas de organizar y almacenar datos en un programa.

  Ejemplo en C#:

```
// Arreglo de enteros
int[] edades = {25, 30, 35};
// Lista de cadenas de texto
List<string> nombres = new List<string>() {"Juan", "María", "Pedro"};

// Diccionario de cadenas de texto y enteros
Dictionary<string, int> edadPorNombre = new Dictionary<string, int>()
{
```

```
    {"Juan", 25},
    {"María", 30},
    {"Pedro", 35}
};
```

En este ejemplo, se utilizan tres estructuras de datos diferentes:

- Un arreglo de enteros edades para almacenar las edades de varias personas.

- Una lista de cadenas de texto nombres para almacenar los nombres de varias personas.

- Un diccionario de cadenas de texto y enteros edadPorNombre para almacenar las edades de varias personas, donde la clave es el nombre de la persona y el valor es su edad.

- ✓ **Código fuente:** El código fuente es el texto que se escribe en un lenguaje de programación para crear un programa.

  Ejemplo en C#:

```
using System;

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hola, mundo!");
    }
}
```

En este ejemplo, se escribe un programa en C# que imprime el mensaje “Hola, ¡mundo!" en la consola. El código fuente incluye la declaración de la clase Program, el método Main y la instrucción Console.WriteLine.

Programación orientada a objetos

- ✓ **Elementos:** La programación orientada a objetos (POO) es un paradigma de programación que se basa en la creación de objetos que representan entidades del mundo real. Estos objetos tienen propiedades y comportamientos que se definen mediante una serie de elementos que se describen a continuación:

  Elementos de la POO

  - Clases: plantillas o moldes que definen las propiedades y comportamientos de un objeto.

  - Objetos: instancias de una clase que tienen propiedades y comportamientos definidos en la clase.

  - Propiedades (Atributos): características de un objeto que se almacenan en variables.

  - Métodos: acciones que un objeto puede realizar.

  - Herencia: capacidad de una clase para heredar las propiedades y comportamientos de otra clase.

  - Polimorfismo: capacidad de un objeto para tomar diferentes formas.

  - Encapsulación: capacidad de un objeto para ocultar sus propiedades y comportamientos internos y solo exponer una interfaz pública.

  - Abstracción: capacidad de un objeto para representar conceptos y entidades del mundo real de manera simplificada.

  Ejemplos y Conceptos Adicionales

  - Algoritmos, pseudocódigos, flujogramas, estructuras de control, tipos de variables, estructuras de datos, funciones y librerías básicas, documentación de programas, código fuente.

- ✓ **Características:**

- **Encapsulación:** La encapsulación es la capacidad de un objeto para ocultar sus propiedades y comportamientos internos y solo exponer una interfaz pública. Esto ayuda a proteger la integridad de los datos y a mejorar la seguridad.
- **Abstracción:** La abstracción es la capacidad de un objeto para representar conceptos y entidades del mundo real de manera simplificada. Esto ayuda a reducir la complejidad y a mejorar la comprensión del sistema.
- **Herencia:** La herencia es la capacidad de una clase para heredar las propiedades y comportamientos de otra clase. La clase que hereda se llama subclase, y la clase de la que se hereda se llama superclase.
- **Polimorfismo:** El polimorfismo es la capacidad de un objeto para tomar diferentes formas. Esto se logra mediante la sobrecarga de métodos o la sobreescritura de métodos.
- **Composición:** La composición es la capacidad de un objeto para estar compuesto por otros objetos. Esto permite crear objetos complejos a partir de objetos más simples.
- **Interfaz:** La interfaz es la capacidad de un objeto para exponer una serie de métodos y propiedades que pueden ser utilizados por otros objetos.
- **Modularidad:** La modularidad es la capacidad de un sistema para estar dividido en módulos independientes que pueden ser desarrollados y mantenidos de manera separada.
- **Reutilización de código:** La reutilización de código es la capacidad de un sistema para reutilizar código existente en lugar de tener que escribirlo nuevamente.

✓ **Propiedades y ventajas de la programación orientada a objetos:**

⇨ **Propiedades**

- **Modularidad:** La POO permite dividir un sistema en módulos independientes, lo que facilita el desarrollo, mantenimiento y reutilización de código.
- **Reutilización de código:** La POO permite reutilizar código existente en lugar de tener que escribirlo nuevamente, lo que ahorra tiempo y recursos.
- **Flexibilidad:** La POO permite crear objetos que pueden ser utilizados en diferentes contextos y situaciones.
- **Escalabilidad:** La POO permite crear sistemas que pueden ser fácilmente escalados y modificados para adaptarse a nuevas necesidades.
- **Mantenimiento:** La POO permite crear sistemas que son fáciles de mantener y modificar, ya que los cambios se pueden realizar en un módulo específico sin afectar al resto del sistema.

⇨ **Ventajas de la POO:**

- **Mejora la organización del código**
- La POO ayuda a organizar el código de manera lógica y estructurada, lo que facilita su comprensión y mantenimiento.
- **Reduce la complejidad:** La POO ayuda a reducir la complejidad de los sistemas al dividirlos en módulos independientes y reutilizables.
- **Mejora la reutilización de código:** La POO permite reutilizar código existente en lugar de tener que escribirlo nuevamente, lo que ahorra tiempo y recursos.

- **Facilita el mantenimiento:** La POO permite crear sistemas que son fáciles de mantener y modificar, ya que los cambios se pueden realizar en un módulo específico sin afectar al resto del sistema.
- **Mejora la escalabilidad:** La POO permite crear sistemas que pueden ser fácilmente escalados y modificados para adaptarse a nuevas necesidades.
- **Facilita la colaboración:** La POO permite que varios desarrolladores trabajen en diferentes módulos del sistema de manera independiente, lo que facilita la colaboración y reduce el tiempo de desarrollo.
- **Mejora la seguridad:** La POO permite crear sistemas que son más seguros, ya que los objetos y módulos pueden ser diseñados para proteger la integridad de los datos y prevenir accesos no autorizados.

Estructuras de control

Son instrucciones en un programa que permiten alterar el flujo de ejecución en función de ciertas condiciones o repeticiones. Ayudan a que un programa tome decisiones, repita ejecutar acciones o bloques de código de manera condicional. Por ejemplo, en Dev C++ se usan:

- **if, else, while, for, etc.,** se utilizan para controlar el flujo del programa.

Clases y funciones: Las clases son plantillas para crear objetos (instancias) en programación orientada a objetos (OOP), y las funciones son bloques de código reutilizables que realizan tareas específicas.

- C++ soporta la Programación Orientada a Objetos (OOP), permitiendo el uso de clases, constructores y métodos.

Librerías: Son colecciones de funciones, clases y otros recursos que permiten realizar tareas comunes sin necesidad de escribir el código desde cero.

- Se pueden usar librerías estándar, como **math, stdio.h, conio.h**, para realizar tareas matemáticas comunes

Código fuente: Es el conjunto de instrucciones escritas en un lenguaje de programación que define el comportamiento de un programa.

Desarrollo de programas: Es el proceso completo de crear un software, desde la planificación, escritura de código, pruebas y documentación.

Documentación de programas: Es la descripción detallada del funcionamiento de un programa, incluyendo la explicación del código, las funciones, la estructura de datos y cómo se usa el software.

- Los comentarios en C++ pueden explicar el **propósito del código**, lo que es **útil** para la **comprensión** y **mantenimiento** del programa.

Análisis estructurado de sistemas

El análisis estructurado de sistemas es una metodología utilizada para comprender y modelar sistemas informáticos. Se basa en una serie de principios y técnicas que permiten representar de manera clara los procesos y flujos de información dentro de un sistema.

Objetivos, entradas, salidas y fases:

El principal objetivo del análisis estructurado es descomponer un sistema en partes más manejables para facilitar su comprensión y desarrollo. Para ello, se consideran:

- **Entradas:** Datos o información que ingresa al sistema.
- **Salidas:** Información procesada que genera el sistema.
- **Fases:** Etapas del análisis estructurado, que incluyen la identificación de requerimientos, modelado del sistema y validación del diseño.

Modelización de funciones y procesos:

Para representar gráficamente el comportamiento del sistema, se utilizan herramientas como diagramas de flujo, diagramas de contexto y diagramas de procesos. Estos permiten visualizar la interacción entre los distintos componentes del sistema.

Figura 3

Modelización de funciones y procesos

Nota: Visualización de la interacción entre los distintos componentes del sistema.

Modelización de datos y análisis entidad-relación

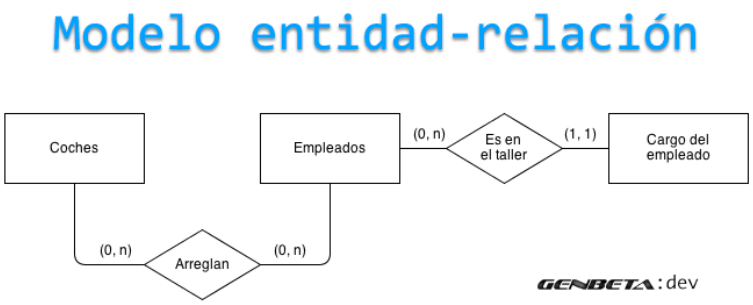
Modelo conceptual: Se usa para describir la estructura de la información sin detalles técnicos.

Análisis entidad-relación: Representa los datos y sus relaciones mediante entidades y atributos.

Diccionario de datos: Un repositorio con definiciones de los elementos de datos utilizados en el sistema.

Figura 4

Modelización de datos y análisis entidad-relación



Nota: Visualización de un modelo de entidad-relación.

Programación con sistemas gestores de bases de datos relacionales:

En esta etapa, se emplean sistemas como MySQL, PostgreSQL o SQL Server para gestionar y manipular la información almacenada en bases de datos relacionales. Se utilizan lenguajes como SQL para realizar consultas, inserciones y modificaciones de datos.

Figura 5

Programación con sistemas gestores de bases de datos relacionales

Nota: Visualización de sistemas de gestores de bases de datos relacionales

Introducción a los sistemas de información

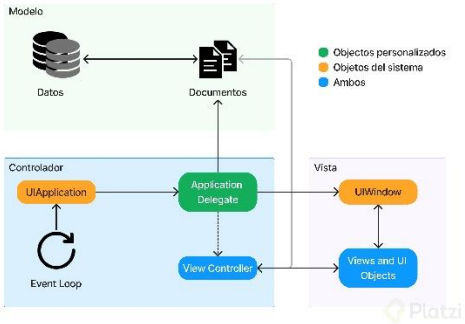
Los sistemas de información son herramientas esenciales en la informática, ya que permiten la gestión eficiente de datos y procesos dentro de una organización.

Metodologías de desarrollo y ciclo de vida de una aplicación:

El desarrollo de sistemas de información sigue metodologías como el modelo en cascada, modelo ágil o desarrollo incremental. Estas metodologías estructuran el proceso de creación de software en fases como análisis, diseño, implementación, pruebas y mantenimiento.

Figura 6

Metodologías de desarrollo y ciclo de vida de una aplicación



Nota: Se observa el desarrollo de los sistemas de información.

Análisis de necesidad y estudios de viabilidad:

Antes de iniciar un proyecto, se realiza un estudio para determinar si el sistema es viable técnica y económicamente. Esto implica evaluar recursos, costos y beneficios esperados.

Análisis de requisitos y gestión de proyectos informáticos:

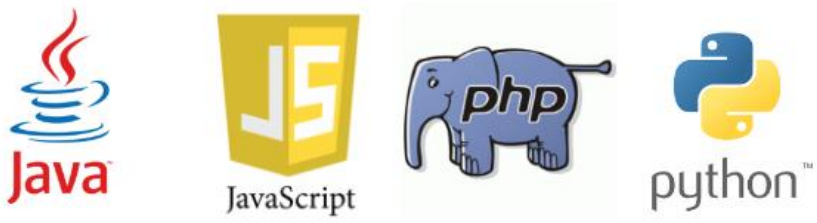
Se identifican los requisitos del sistema mediante técnicas como entrevistas y encuestas. Luego, se gestionan los recursos, tiempos y tareas del proyecto para garantizar su éxito.

Programación con bases de datos relacionales y lenguajes orientados a cliente-servidor:

Los sistemas de información utilizan bases de datos relacionales para almacenar datos de manera estructurada y accesible. Además, se desarrollan aplicaciones cliente-servidor que permiten la interacción entre usuarios y servidores a través de protocolos de comunicación y tecnologías como Java, PHP, y JavaScript.

Figura 7

Programación con bases de datos relacionales y lenguajes orientados a cliente-servidor



Nota: Se observa servidores a través de protocolos de comunicación y tecnologías como Java, PHP, y JavaScript.

Herramientas de desarrollo

Las herramientas de desarrollo son software que ayudan en la creación, diseño, prueba y mantenimiento de aplicaciones. Estas herramientas simplifican la creación de interfaces de usuario, generación de informes y desarrollo de aplicaciones. A continuación, te explico cada una de estas herramientas con más detalle:

1. **Generadores de pantallas (Interfaces de usuario)**

Un generador de pantallas es una herramienta para crear interfaces gráficas de usuario (GUI) sin escribir mucho código. Permite arrastrar y soltar elementos para diseñar aplicaciones, facilitando el proceso a quienes tienen poca experiencia en diseño.

Ejemplos de generadores de pantalla:

Visual Studio: Ofrece herramientas como el Windows Forms Designer o WPF para crear interfaces gráficas en aplicaciones de escritorio en C# o VB.NET.

Delphi: Tiene un generador de pantallas visuales que permite crear aplicaciones de escritorio en Pascal rápidamente.

Qt Designer: Una herramienta para diseñar interfaces gráficas en C++ utilizando la librería Qt.

Ejemplo de uso:

En un generador de pantallas, puedes arrastrar un botón y un cuadro de texto sobre una "pantalla" en blanco. Luego, puedes asociar eventos, como el clic del botón, a una acción (por ejemplo, mostrar el texto escrito en el cuadro de texto). La herramienta genera el código necesario automáticamente.

2. Generadores de informes

Los generadores de informes son herramientas que crean informes automáticamente a partir de datos, sin programar la lógica desde cero. Permiten conectar la base de datos de una aplicación y diseñar informes visuales.

Ofrecen interfaces visuales para elegir los campos de datos y el formato de visualización.

Ejemplos de generadores de informes:

Crystal Reports: Una de las herramientas más populares para la creación de informes complejos a partir de bases de datos. Permite diseñar informes con diferentes tipos de visualización (tablas, gráficos, etc.).

JasperReports: Una herramienta basada en Java que permite crear informes en diferentes formatos (PDF, HTML, Excel).

SQL Server Reporting Services (SSRS): una plataforma de Microsoft para crear, administrar y entregar informes interactivos y basados en datos a los usuarios.

Ejemplo de uso:

Si tienes una base de datos con una tabla de ventas, puedes usar un generador de informes como Crystal Reports para crear un informe que muestre las ventas totales por cada vendedor, de forma visual y estructurada.

3. Generadores de consultas

Un generador de consultas ayuda a crear consultas SQL sin escribir manualmente. Permite seleccionar visualmente tablas, campos y condiciones, generando automáticamente el código SQL. Es útil para quienes no tienen experiencia en SQL o necesitan generar consultas complejas rápidamente.

Ejemplos de generadores de consultas:

SQL Server Management Studio (SSMS) : La herramienta de Microsoft para trabajar con SQL Server incluye un generador de consultas visual que permite crear consultas sin escribir código manualmente.

Toad para MySQL/Oracle: Herramientas que permiten generar consultas SQL de manera visual.

MySQL Workbench: Ofrece un generador de consultas SQL visual para bases de datos MySQL.

Ejemplo de uso:

Si tienes una base de datos de empleados y deseas obtener todos los empleados que trabajan en el departamento de "Ventas", un generador de consultas puede permitirte seleccionar la tabla "empleados", elegir los campos "nombre" y "departamento", y aplicar un filtro con "departamento = 'Ventas'" para generar la consulta SQL automáticamente.

4. Generador de aplicaciones

Un generador de aplicaciones es una herramienta que permite crear aplicaciones completas sin escribir todo el código desde cero. Incluye módulos predefinidos para tareas comunes, facilitando el trabajo a desarrolladores y usuarios sin experiencia técnica gracias a interfaces de "arrastrar y soltar".

Ejemplos de generadores de aplicaciones:

AppGyver: Plataforma para crear aplicaciones móviles sin código (no-code), con una interfaz visual.

OutSystems: Un entorno de desarrollo rápido de aplicaciones que permite crear aplicaciones empresariales a gran escala sin necesidad de escribir mucho código.

FileMaker: Herramienta de creación de aplicaciones empresariales que facilita la creación de soluciones personalizadas, como bases de datos, con poco o ningún código.

Ejemplo de uso:

Usando una herramienta como OutSystems, puedes crear una aplicación de gestión de empleados arrastrando módulos predefinidos, como formularios para agregar empleados, listas para ver todos los empleados, y pantallas para editar la información de los empleados, todo sin necesidad de escribir mucho código.

Beneficios:

Ahorro de tiempo: Permiten desarrollar aplicaciones y sistemas mucho más rápidos, ya que automatizan tareas complejas o repetitivas.

Facilidad de uso: Muchas de estas herramientas están diseñadas para que personas con pocos conocimientos de programación puedan usarlas, facilitando la creación de aplicaciones e informes.

Reducción de errores: Al generar automáticamente el código necesario, se reducen los errores humanos que podrían ocurrir al escribirlo manualmente.

Flexibilidad: Aunque estas herramientas facilitan el trabajo, aún permiten que los desarrolladores agreguen código personalizado cuando sea necesario para adaptar las soluciones a necesidades específicas.

Generación y desarrollo de aplicaciones con herramientas case

Las herramientas CASE (Computer-Aided Software Engineering) son software que ayudan en el desarrollo de aplicaciones de software, mejorando el ciclo de vida desde la planificación hasta el mantenimiento. Se explicará la generación y desarrollo de aplicaciones con estas herramientas, destacando características, estructura, generación de código, documentación y más.

1. Características de las Herramientas CASE

Las herramientas CASE tienen características diseñadas para optimizar y hacer más eficiente el proceso de desarrollo de software. Algunas de sus características más comunes incluyen:

Automatización de tareas repetitivas: Ayudan a automatizar muchas de las tareas involucradas en el desarrollo, como la creación de diagramas, la generación de código o la creación de documentación.

Soporte para Modelado: Ofrecen herramientas para diseñar diagramas de flujo, diagramas de casos de uso, diagramas de clases, diagramas de actividad y otros modelos que representan el sistema a desarrollar.

Generación de código: Pueden generar automáticamente código a partir de los modelos desarrollados, reduciendo el tiempo y los errores humanos al escribir código manualmente.

Manejo de la documentación: Ayudan a generar documentación del software y proporcionar informes sobre el desarrollo y el estado del proyecto.

Control de versiones y seguimiento de cambios: Muchas herramientas CASE permiten gestionar versiones del proyecto, lo que facilita la actualización y el mantenimiento de los sistemas.

2. **Estructura de las Herramientas CASE**

Las herramientas CASE están divididas en varios módulos, cada uno para una fase del ciclo de vida del software. Incluyen:

Módulo de análisis y diseño: crea modelos del sistema.

Módulo de generación de código: genera código fuente automáticamente.

Módulo de pruebas: gestiona pruebas automatizadas.

Módulo de documentación: genera documentación técnica.

Módulo de gestión de proyectos: planifica y sigue actividades del proyecto.

3. Actualización y Mantenimiento del Proyecto

Las herramientas CASE permiten actualizar y mantener el software a lo largo de su ciclo de vida. Algunas de las funcionalidades relacionadas con la actualización incluyen:

Refactorización automática: Permite actualizar y mejorar el código de manera automática sin alterar su funcionalidad.

Control de versiones: Las herramientas CASE suelen incluir un sistema para gestionar las versiones del software y realizar un seguimiento de los cambios a lo largo del tiempo.

Mantenimiento de modelos: Si se realizan cambios en los requisitos o el diseño, las herramientas CASE permiten actualizar los diagramas y modelos, lo que actualiza automáticamente la documentación y el código correspondiente.

4. Generación de Código

La generación de código es una de las funcionalidades clave de las herramientas CASE. Estas herramientas permiten generar código de forma automática a partir de los modelos creados, lo que ayuda a reducir la cantidad de código repetitivo que debe escribirse manualmente. Esto también ayuda a garantizar que el código sea consistente y esté alineado con los requisitos del sistema.

Generación desde diagramas: Por ejemplo, si se crea un diagrama de clases, la herramienta CASE puede generar las clases correspondientes en el lenguaje de programación deseado (por ejemplo, C++, Java, Python).

Generación de código de bases de datos: También pueden generar código para la creación de bases de datos y la manipulación de datos (por ejemplo, sentencias SQL).

5. Documentación generada

Las herramientas CASE también son esenciales para la creación y mantenimiento de la documentación de software. Pueden generar documentación técnica y de usuario de forma automática y actualizada en función del estado del proyecto.

Documentación técnica: A partir de los diagramas y el código generado, las herramientas CASE pueden crear documentación sobre el diseño, las clases, los métodos, las relaciones entre componentes y las decisiones arquitectónicas.

Manuales de usuario: Las herramientas CASE también pueden generar manuales de usuario para explicar cómo interactuar con el software, con instrucciones sobre cómo usar las funcionalidades.

Manuales de programador: Además de los manuales de usuario, pueden generar documentación dirigida a los programadores, que describe la estructura del código, las clases y métodos, y cómo se puede modificar o extender el software.

6. Tutoriales y Reportes de Desarrollo

Las herramientas CASE suelen incluir tutoriales integrados e informes de desarrollo que ayudan a los desarrolladores a entender cómo usar la herramienta ya seguir el progreso del proyecto. Algunos ejemplos hijo:

Tutoriales interactivos: Proporcionan guías paso a paso para aprender a usar la herramienta, lo que puede ser útil para los nuevos usuarios.

Reportes de desarrollo: Incluyen información sobre el estado del proyecto, los componentes desarrollados, las tareas completadas, las pruebas realizadas, etc. Estos informes ayudan en la gestión del proyecto y permiten a los desarrolladores y líderes de equipo mantenerse al tanto de los avances.

7. Ventajas de Usar Herramientas CASE

Aumento de la productividad: La automatización de tareas repetitivas como la generación de código y la documentación reducen significativamente el tiempo de desarrollo.

Mejora de la calidad: Al generar código automáticamente desde modelos bien definidos, las herramientas CASE reducen la probabilidad de errores humanos en la codificación.

Facilita la colaboración: Proporcionan una representación visual de la arquitectura y los componentes del sistema, lo que facilita la colaboración entre los miembros del equipo de desarrollo.

Mejora la gestión del proyecto: Al integrar la planificación, el diseño y la gestión de versiones en una sola herramienta, los equipos pueden llevar un control más efectivo del proyecto.

Ejemplos de herramientas CASE:

Algunas de las herramientas CASE más utilizadas en el desarrollo de software incluyen:

IBM Rational Rose: una herramienta de modelado que permite crear diagramas UML y generar código.

Enterprise Architect: Utilizada para el modelado y diseño de sistemas, incluye funciones de generación de código, pruebas y documentación.

Visual Paradigm: Ofrece soporte para el modelado UML, la generación de código y la creación de diagramas de bases de datos