

Introducción a la Ingeniería del Software

Código del curso:
SOFT-09

METODOLOGÍA DE EVALUACIÓN

Actividades de aprendizaje	Porcentaje
Propuesta conceptual de software (1)	40%
Prácticas (2)	30%
Exposiciones (2)	30%
Total	100%

Proyecto 40%

Avance 1	5%
Avance 2	10%
Avance Final	15%
Exposición(Defensa)	10%

Prácticas 30%

Práctica 1	15%
Práctica 2	15%

Exposiciones 30%

Exposición 1	15%
Exposición 2	15%

AGENDA

MÓDULO 1. Fundamentos de ingeniería del software

1. Historia y evolución de la ingeniería del software

2. Definición de procesos de software y ciclos de vida

3. Estructura y arquitectura de software.

4. Paradigmas de programación

5. Evaluación y mejora de procesos de software

6. Ética y profesionalismo en la ingeniería del software.

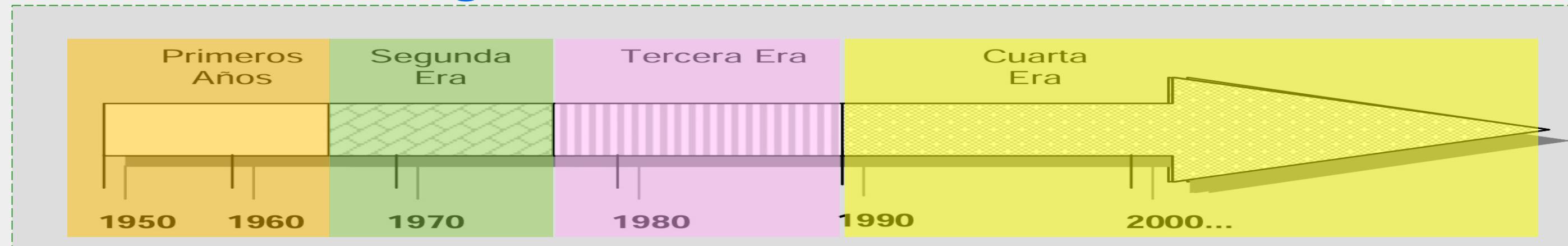
Historia y evolución de la ingeniería del software

El término ingeniería del software apareció por primera vez en la década de 1950 y principios de los años 1960.

Los programadores siempre habían sabido sobre ingenieros civiles, ingenieros eléctricos y los ingenieros de computadores y se preguntaban ¿qué podría significar la ingeniería para el software?.



Línea de tiempo detallada de los hitos más importantes en la evolución de la ingeniería de software



Año	Evento
1945	El matemático británico Alan Turing escribe un artículo sobre la posibilidad de crear máquinas que puedan realizar cualquier tarea computable.
1951	Grace Hopper desarrolla el primer compilador de programación de alto nivel, el A-0.
1956	John Backus y su equipo desarrollan el primer lenguaje de programación de alto nivel, el Fortran.
1968	Se celebra la primera conferencia de ingeniería de software en Garmisch, Alemania.
1970	Se desarrolla el lenguaje de programación Pascal, diseñado para enseñar programación estructurada.
1971	Se crea el primer sistema operativo de la historia, el Unix, desarrollado por Dennis Ritchie y Ken Thompson.
1972	Se publica el libro "The Art of Computer Programming" de Donald Knuth, un referente fundamental en la informática teórica.
1975	Se crea el lenguaje de programación C, desarrollado por Dennis Ritchie en los laboratorios Bell.
1986	Se publica el primer estándar de calidad de software, el IEEE 730, que establece las directrices para la planificación y control de calidad del software.

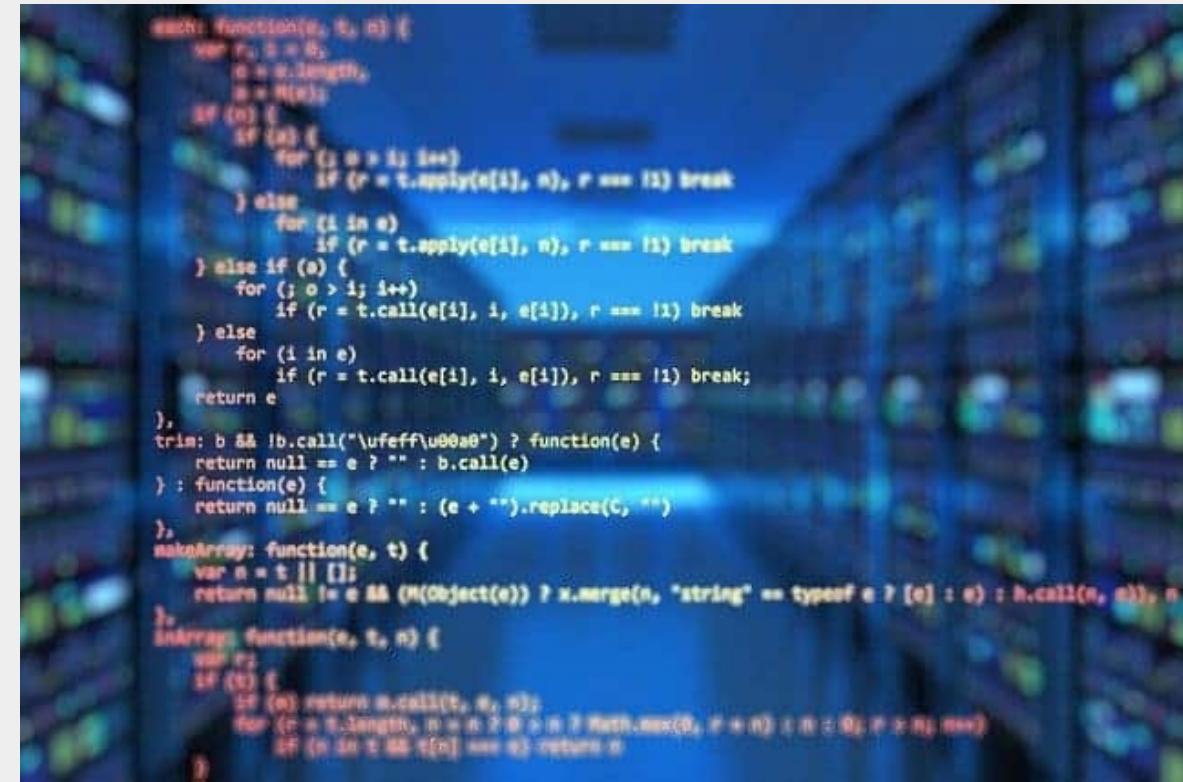
1987	Se publica el estándar ISO 9001, que establece los requisitos para un sistema de gestión de calidad en una organización.
1995	Se publica el estándar ISO/IEC 12207, que establece los procesos del ciclo de vida del software.
2001	Un grupo de programadores crea el Manifiesto Ágil, que establece los valores y principios para el desarrollo de software ágil.
2006	Se publica el estándar ISO/IEC 15504, que evalúa la capacidad de los procesos de software y establece los requisitos para la mejora continua.
2014	Se publica la versión 3.0 del CMMI, un modelo de gestión de calidad y mejora continua de procesos en ingeniería de software.

¿Qué estudia la ingeniería de software?

1. Procesos para crear software
2. Entender lo que necesita el software
3. Escribir el código del programa
4. Revisar que todo funcione bien
5. Mantener y mejorar el software
6. Proteger el software de ataques
7. Hacer programas fáciles de usar
8. Usar inteligencia artificial en el desarrollo

¿Qué es un software?

Es un conjunto de instrucciones y datos que permiten a una computadora realizar tareas específicas.



```
each: function(e, t, n) {
    var r, i = 0;
    e = e.length;
    n = n || [];
    if (n) {
        if (e) {
            for (i = 0; i < e; i++) {
                if (r = t.apply(n[i], e), r === i) break;
            } else
                for (i in e)
                    if (r = t.apply(e[i], e), r === i) break;
        } else if (e) {
            for (; i > e; i++)
                if (r = t.call(e[i], i, e[i]), r === i) break;
        } else
            for (i in e)
                if (r = t.call(e[i], i, e[i]), r === i) break;
        return e;
    },
    trim: b =&& b.call("\uffff\ufe0f") ? function(e) {
        return null == e ? "" : b.call(e)
    } : function(e) {
        return null == e ? "" : (e + "").replace(c, "")
    },
    makeArray: function(e, t) {
        var n = t || [];
        return null != e && (N(Object(e)) ? x.merge(n, "string" == typeof e ? [e] : e) : b.call(n, e)), n
    },
    isArray: function(e, t, n) {
        var r;
        if (t) {
            if (n) return b.call(t, n, n);
            for (r = 0; r < e.length; r++) if (b.call(n, r + 0) !== 0 || r > n) return n;
        }
    }
},
```

Lenguaje de BAJO NIVEL

```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER PAGE  2

C000          ORG    ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START  LDS    #STACK
*****
* FUNCTION: INITA - Initialize ACIA
* INPUT: none
* OUTPUT: none
* CALLS: none
* DESTROYS: acc A

0013        RESETA EQU    %00010011
0011        CTLREG EQU    %00010001

C003 86 13  INITA   LDA A #RESETA  RESET ACIA
C005 B7 80 04      STA A ACIA
C008 86 11      LDA A #CTLREG  SET 8 BITS AND 2 STOP
C00A B7 80 04      STA A ACIA

C00D 7E C0 F1      JMP     SIGNON  GO TO START OF MONITOR
*****
* FUNCTION: INCH - Input character
* INPUT: none
* OUTPUT: char in acc A
* DESTROYS: acc A
* CALLS: none
* DESCRIPTION: Gets 1 character from terminal

C010 B6 80 04  INCH    LDA A ACIA    GET STATUS
C013 47          ASR A     SHIFT RDRF FLAG INTO CARRY
C014 24 FA          BCC    INCH    RECEIVE NOT READY
C016 B6 80 05      LDA A ACIA+1  GET CHAR
C019 84 7F          AND A #$7F    MASK PARITY
C01B 7E C0 79      JMP     OUTCH   ECHO & RTS

*****
* FUNCTION: INHEX - INPUT HEX DIGIT
* INPUT: none
* OUTPUT: Digit in acc A
* CALLS: INCH
* DESTROYS: acc A
* Returns to monitor if not HEX input

C01E 8D F0  INHEX   BSR     INCH    GET A CHAR
C020 81 30      CMP A #'0    ZERO
C022 2B 11      BMI     HEXERR  NOT HEX
C024 81 39      CMP A #'9    NINE
C026 2F 0A      BLE     HEXRTS  GOOD HEX
C028 81 41      CMP A #'A    HEXERR
C02A 2B 09      BMI     HEXERR  NOT HEX
C02C 81 46      CMP A #'F    HEXERR
C02E 2E 05      BGT     HEXERR
C030 80 07      SUB A #7    FIX A-F
C032 84 0F      HEXRTS  AND A #$0F  CONVERT ASCII TO DIGIT
C034 39          RTS

C035 7E C0 AF  HEXERR  JMP     CTRL   RETURN TO CONTROL LOOP
```

Lenguaje de ALTO NIVEL

python

```
# Programa para registrar nombre y edad
# Solicitar datos al usuario
nombre = input("Ingrese su nombre: ")
edad = input("Ingrese su edad: ")

# Mostrar mensaje de confirmación
print(f"\nRegistro completado.")
print(f"Nombre: {nombre}")
print(f"Edad: {edad} años")
```

Ingrese su nombre: Ana

Ingrese su edad: 25

Registro completado.

Nombre: Ana

Edad: 25 años

¿Qué tipos de software hay?

Abarca toda la parte del sistema operativo y el procesamiento de elementos de hardware como los discos, los puertos, las pantallas, los teclados, la memoria, etc. Ofrece al usuario una interfaz gráfica amigable y las funcionalidades que necesita: S.O., servidores, controladores, entre otros.

- Software de sistema



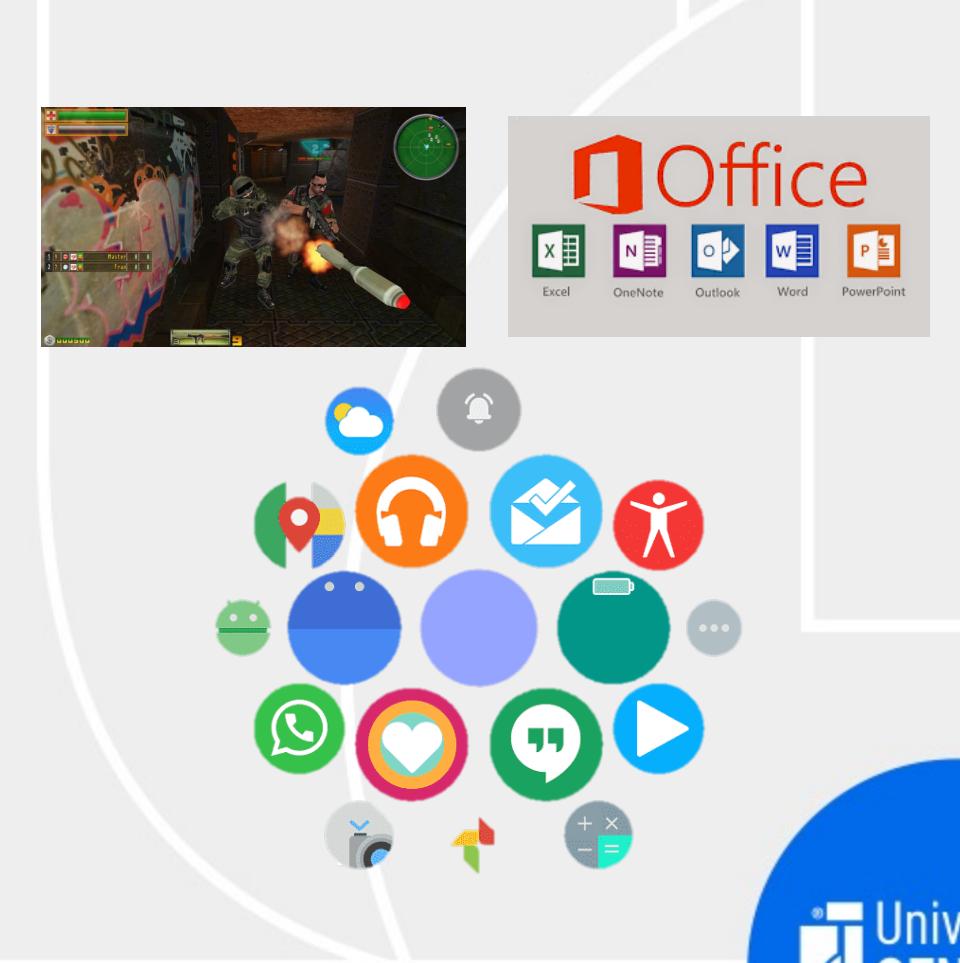
Abarca las herramientas mediante las cuales el programador desarrolla determinados programas de informática, en distintos entornos y lenguajes de programación. En definitiva, utilizando ciertos programas o software, podemos crear más programas: editores de texto, compiladores, , softwares para gestión, entre otros.

- Software de programación



En el caso de este otro tipo de software, se refiere al que permite realizar ciertas actividades concretas. Es decir, el que ha sido programado para ejecutarse en los sistemas operativos o en la nube. Por ejemplo, aplicaciones empresariales, aplicaciones educativas, videojuegos,

- Software de aplicación



Definición de procesos de software y ciclos de vida

Proceso de software

- Es una serie de actividades relacionadas que conduce a la elaboración de un producto de software. Estas actividades pueden incluir el desarrollo de software desde cero en un lenguaje de programación estándar como Java o C

Algunos procesos o metodologías son:

- El modelo en cascada.
- Desarrollo incremental.
- Ingeniería de software orientada a la reutilización.

Ciclo de vida del software

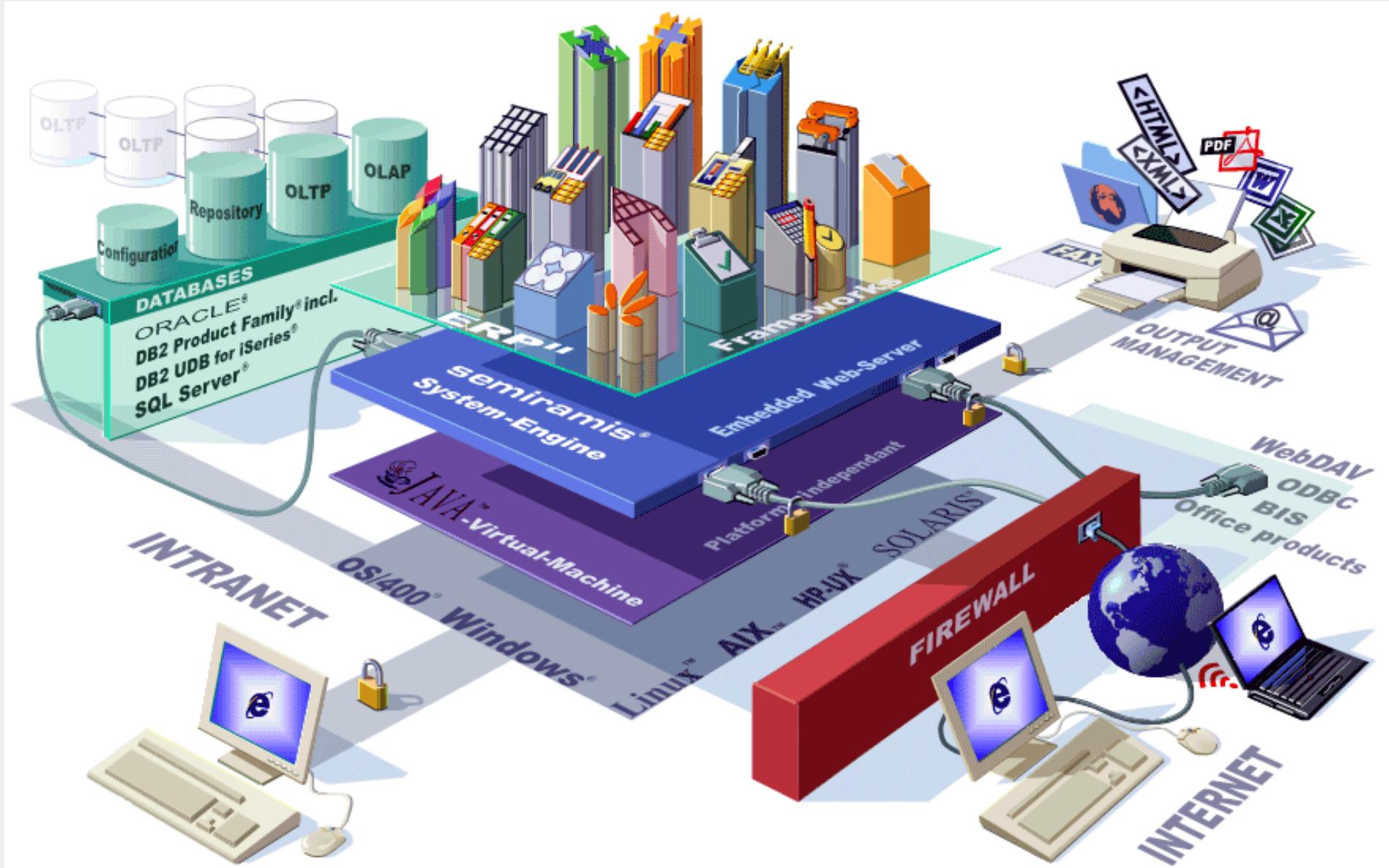


<i>FASES</i>	<i>ACTIVIDADES</i>	<i>ARTEFACTOS</i>
<i>ANÁLISIS</i>	Definición Alcance del proyecto	
	Identificación del negocio	Modelo del negocio
	Toma de Requerimientos	Análisis y realización de casos de uso
	Estudio de Procesos del Negocio	Modelo de procesos y actividades del negocio
	Calendarización del Proyecto	Cronograma del proyecto
<i>DISEÑO</i>	Identificación de la Arquitectura de Software	Modelo de Arquitectura de Software
	Análisis de almacenamiento de la información	Modelo de Bases Datos
	Análisis de objetos	Modelo de Clases y de Patrones
<i>IMPLEMENTACIÓN</i>	Construcción del Software	Producto de Software
<i>PRUEBAS</i>	Descripción técnica de las pruebas	Plan de pruebas
	Selección de las métricas de pruebas	Ánálisis de los resultados de las métricas establecidas
<i>MANTENIMIENTO</i>	Control de cambios	Plan de gestión de cambios
	Calidad del Proyecto	Plan de aseguramiento de la calidad
	Administración del Riesgo	Plan de gestión de riesgo
	Métricas de Mantenimiento	Ánálisis de los resultados de las métricas establecidas

ESTRUCTURA Y ARQUITECTURA DEL SOFTWARE

Es la parte intangible.

Los componentes del Software se refieren al conjunto de operaciones lógicas que hacen posible la funcionalidad del sistema de cómputo.



Lenguajes de programación:

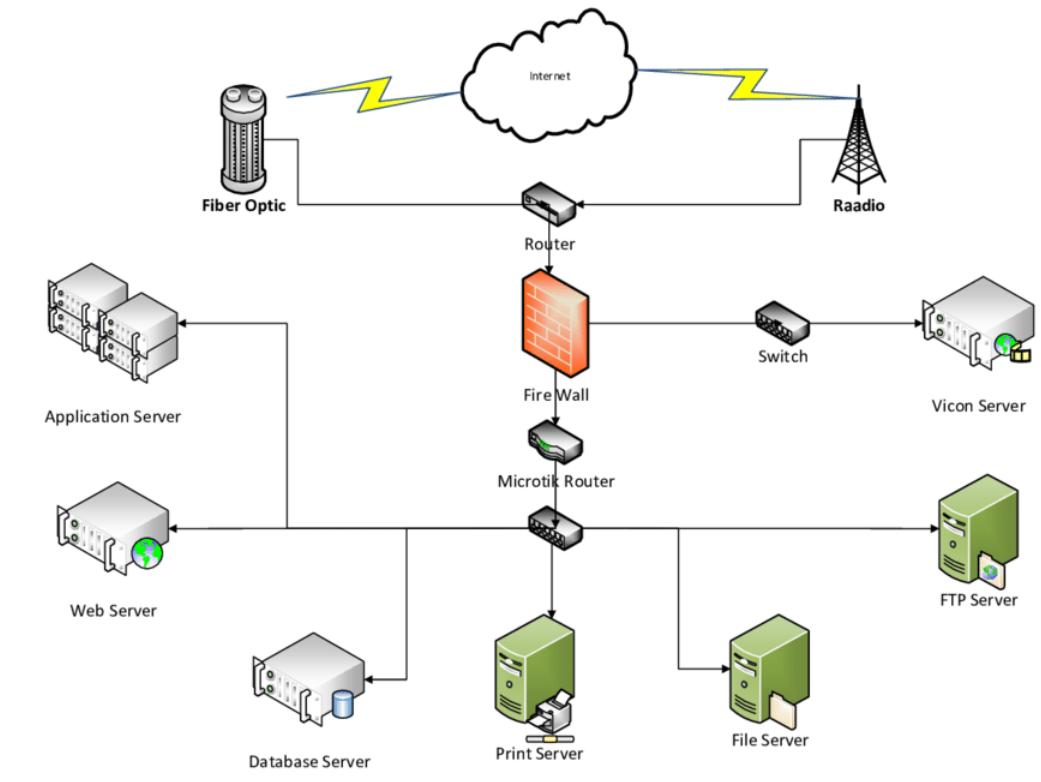
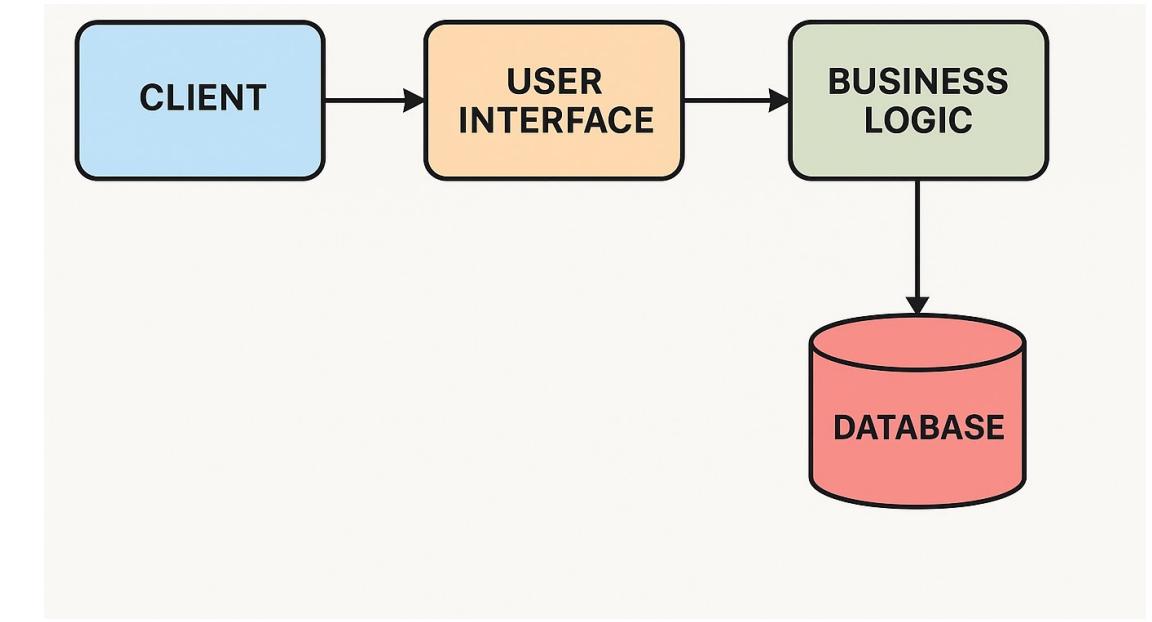
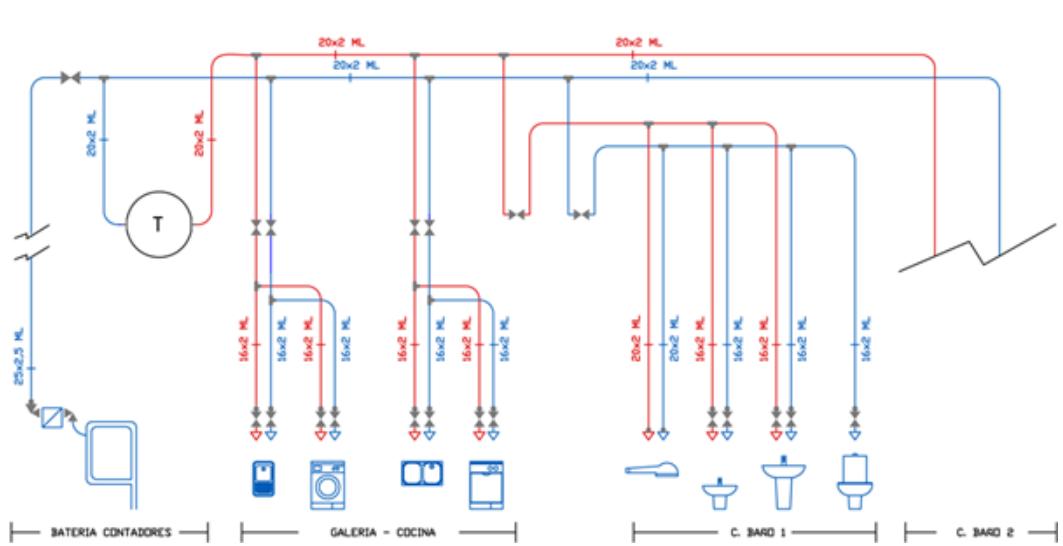
Son instrucciones y reglas que necesitamos para que el desarrollador pueda transmitir órdenes a la computadora.

Sistemas operativos:

Es un programa externo de carácter imprescindible, sin los SO el hardware no podría funcionar.

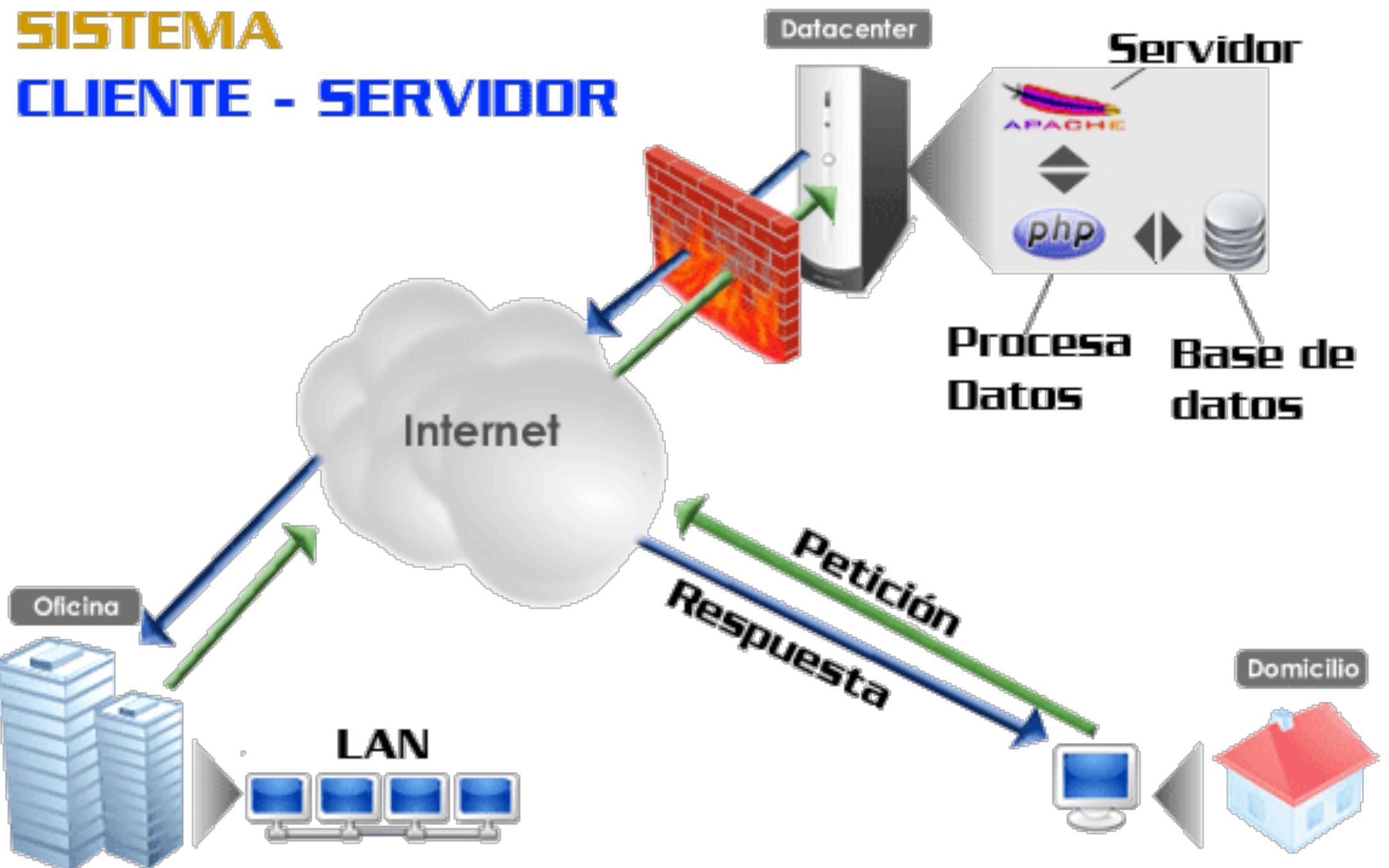
¿Qué es la arquitectura de software?

Se refiere al diseño estructural de los sistemas de software. Es la base sobre la cual se construye y organiza el software. La arquitectura de software se refiere a la estructura general de un sistema de software, incluyendo sus componentes, interacciones y principios rectores. Define el plan maestro tanto del sistema como del proyecto, actuando como puente entre el diseño del sistema y sus requisitos de negocio.



IMPORTANCIA DE LA ARQUITECTURA DEL SOFTWARE

- Mantenible.
- Escalabilidad.
- Rendimiento.
- Reutilización.
- Colaboración en equipo.
- Cumplimiento de Requisitos.



CONCLUSIONES DE LA ARQUITECTURA DEL SOFTWARE

La arquitectura del software es una fase de planificación crítica a la hora de comenzar cualquier proyecto para cualquier pieza de software.

EJERCICIO PARA INICIAR

Recolectar los requerimientos con el usuario:

Tener un Coffee maker

Tipo de café.

Agua

La taza para servir

Filtros

Electricidad

Azúcar, leche, crema.

Paso a paso del algoritmo.

- Conectar el coffee
 - Abrir la tapa del dispensador de agua
 - Llevar el pichel a la pila, abrir el tubo, llenar de agua el pichel hasta 10 tazas, cerrar el tubo.
 - Llevar el pichel con agua y verterlo en el dispensador del agua.
 - Poner el pichel en el disco del coffemaker.
 - Se le pone el filtro de papel en el dispensador del filtro.
 - Ponemos 6 cucharadas de café en polvo. Cerramos la tapa del dispensador.
 - Encender el botón del coffee.
-
- RESULTADO: Café líquido, y lo servimos en la taza de café.



PARADIGMAS DE PROGRAMACIÓN

Son modelos para resolver problemas comunes con nuestro código. Son caminos, guías, reglas, teorías y fundamentos que agilizan nuestro desarrollo y evitan que reinventemos la rueda.



La programación imperativa

Las instrucciones de nuestro programa deben ser bastante explícitas.
El “cómo” realizamos cada paso del algoritmo debe ser muy claro.

```
python  
  
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
pares = []  
  
for n in numeros:  
    if n % 2 == 0:  
        pares.append(n)  
  
print(pares)
```

python

[2, 4, 6, 8, 10]

Ejemplos de lenguajes
de programación
imperativa:
C, C++, Java y Python.

Primer número: 1 → no es par → no se agrega.

Segundo número: 2 → es par → pares.append(2) → lista queda [2].

Luego 4 → se agrega → [2, 4].

Así hasta el final, quedando: [2, 4, 6, 8, 10].

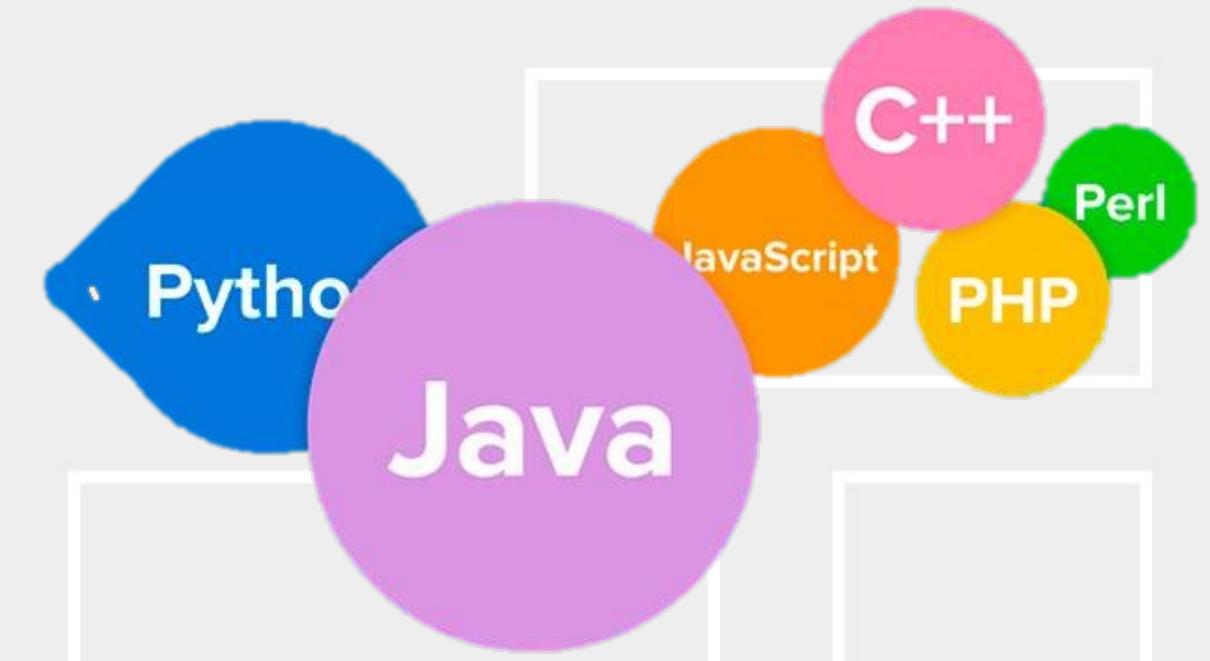
Programación Orientado a Objetos (POO)

```
# Definimos la clase Perro
class Perro:
    def __init__(self, nombre, raza):
        # Atributos del perro
        self.nombre = nombre
        self.raza = raza

    # Método del perro
    def ladrar(self):
        print(f"{self.nombre} está ladranddo: ¡Guau guau!")

# Creamos objetos (instancias) de la clase Perro
perro1 = Perro("Firulais", "Labrador")
perro2 = Perro("Rex", "Pastor Alemán")

# Llamamos al método ladrar de cada objeto
perro1.ladrar()
perro2.ladrar()
```

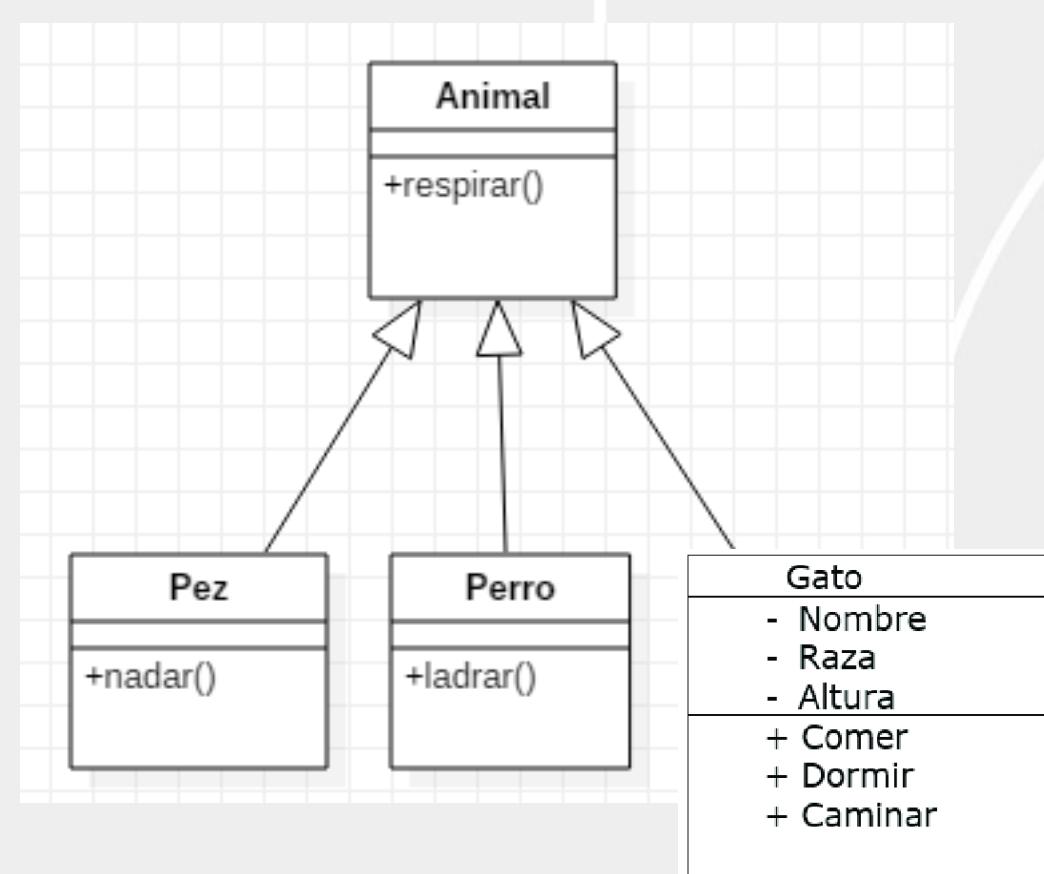


Clase Perro : Es el plano o molde.

Atributos (nombre , raza): Características del perro.

Método ladrar() : Acción que puede hacer el perro.

Objetos (perro1 , perro2): Perros reales creados a partir de la clase.



La programación declarativa

Contrario al paradigma imperativo, la **programación declarativa**, prioriza la claridad del resultado por encima que la claridad del paso a paso. O sea, “dame el resultado que quiero” sin especificarle paso a paso cómo hacerlo.

```
-- Crear una tabla de empleados
CREATE TABLE empleados (
    id INTEGER PRIMARY KEY,
    nombre TEXT,
    departamento TEXT,
    salario INTEGER
);

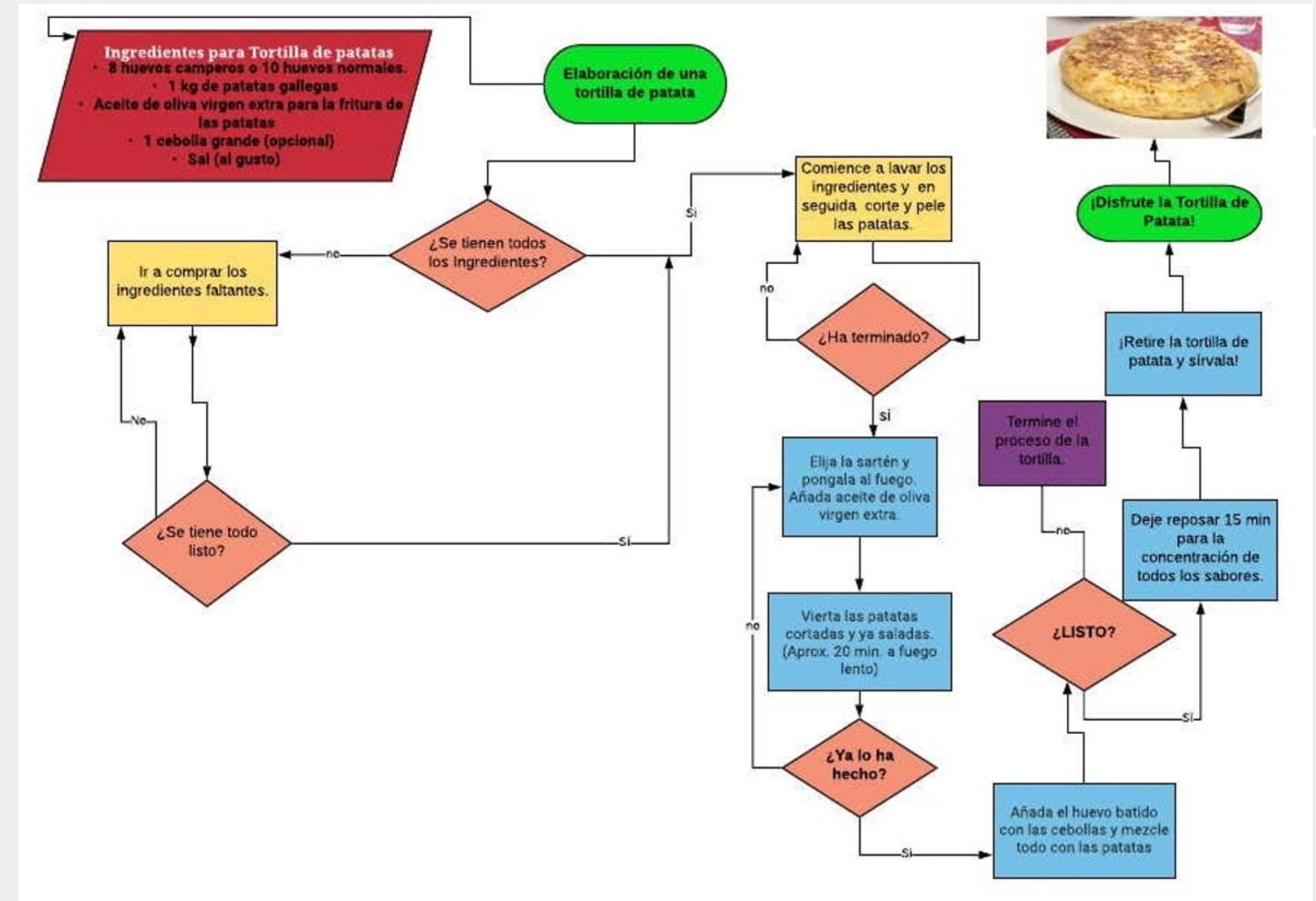
-- Insertar algunos registros
INSERT INTO empleados (id, nombre, departamento, salario) VALUES
(1, 'Ana', 'Ventas', 1200),
(2, 'Luis', 'TI', 1500),
(3, 'Marta', 'Ventas', 1300),
(4, 'Pedro', 'TI', 1800);

-- Consulta declarativa:
-- Quiero obtener todos los empleados del departamento de "Ventas"
SELECT nombre, salario
FROM empleados
WHERE departamento = 'Ventas';
```

Aquí lo declarativo está en que simplemente dices “quiero los empleados de Ventas” (WHERE departamento = 'Ventas'). No le dices al sistema cómo recorrer la tabla, comparar registros o devolverlos, eso lo hace el motor de SQL internamente.

nombre		salario
<hr/>		
Ana		1200
Marta		1300

Programación lógica o declarativa.



La programación es una profesión que requiere dominar la lógica

Ejemplo de Programación Funcional

```
python

# Lista de números
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Programación funcional:
# Quiero obtener solo los números pares y elevarlos al cuadrado
pares_cuadrados = list(
    map(lambda x: x**2, filter(lambda x: x % 2 == 0, numeros))
)

print(pares_cuadrados)
```

[4, 16, 36, 64, 100]

El programa entiende esto:
“De esta lista, filtra los pares y después eleva cada uno al cuadrado”.

PROGRAMACIÓN REACTIVA

La programación reactiva es un paradigma de programación asíncrona orientado al flujo de datos y a la propagación del cambio

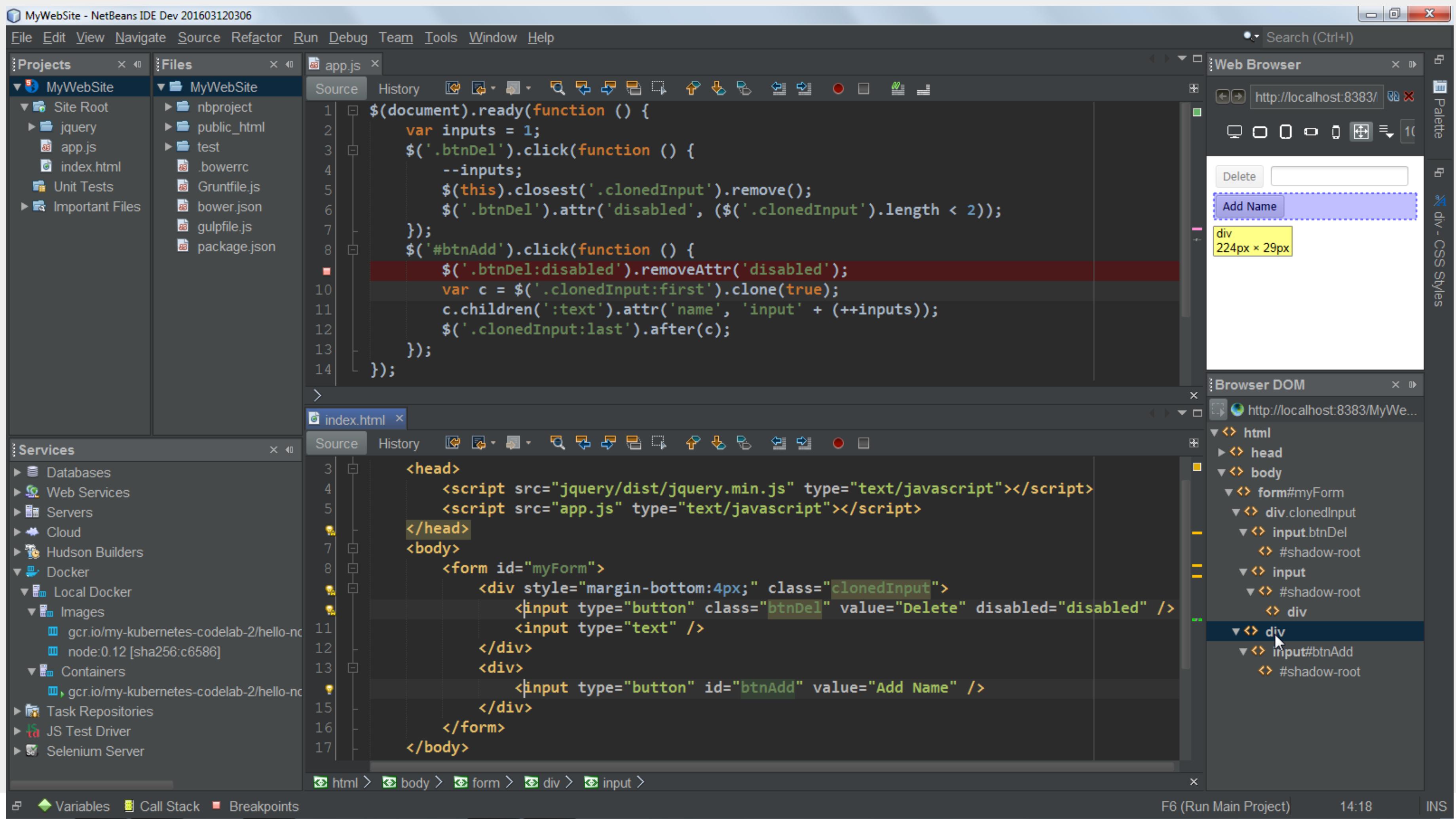
We use ReactiveX



Entorno de desarrollo integrado (IDE)



Lenguaje de programación





free pascal

Main.java

```
1 import java.util.*;
2 class Main {
3     public static void main(String[] args) {
4         Scanner sc = new Scanner(System.in);
5         System.out.println("Escribe la cadena: ");
6         String cadena = sc.nextLine();
7         System.out.println(rot1(cadena));
8         String otraCosa = String.va[1]
9     }
10
11     public static String rot1(str[1]
https://PricklyImportantSystemadministrator.parzbyte.repl.it/
```

java version "1.8.0_31"
Java(TM) SE Runtime Environment (build 1.8.0_31-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.31-b10, mixed mode)

Resaltado de errores

Autocompletado

The screenshot shows a Java code editor with syntax highlighting and code completion. A green arrow points from the word 'va' in line 8 to a tooltip containing code completion options for the String class. Another green arrow points from the word 'rot1' in line 7 to a tooltip showing the definition of the rot1 method. The code itself is a simple program that reads a string from standard input and prints its rotation by one character.

Lenguaje de programación

Sep 2025	Sep 2024	Change	Programming Language	Ratings	Change
1	1		 Python	25.98%	+5.81%
2	2		 C++	8.80%	-1.94%
3	4	▲	 C	8.65%	-0.24%
4	3	▼	 Java	8.35%	-1.09%
5	5		 C#	6.38%	+0.30%
6	6		 JavaScript	3.22%	-0.70%
7	7		 Visual Basic	2.84%	+0.14%
8	8		 Go	2.32%	-0.03%
9	11	▲	 Delphi/Object Pascal	2.26%	+0.49%
10	27	▲	 Perl	2.03%	+1.33%
11	9	▼	 SQL	1.86%	-0.08%

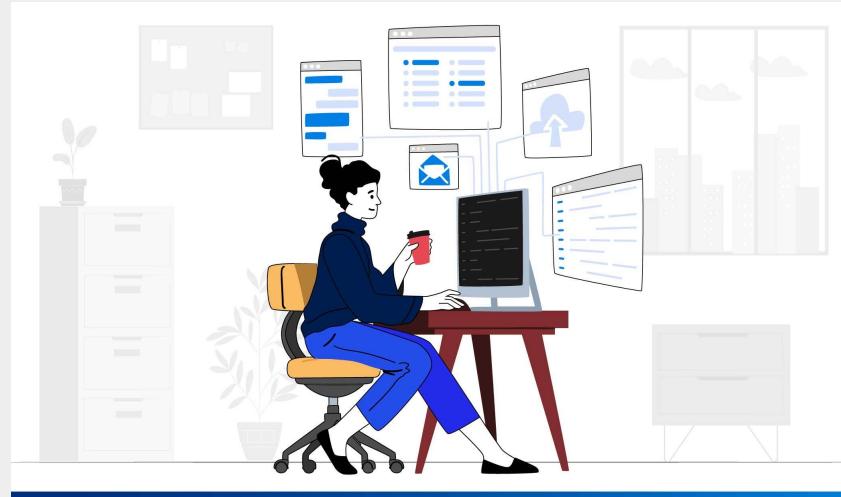


<https://www.tiobe.com/tiobe-index/>

Evaluación y mejora de procesos de software

- La evaluación de software determina la calidad, usabilidad y eficacia de una aplicación o sistema de software.
- Revisar el diseño del software, probar su funcionalidad y rendimiento y evaluar la documentación del software forman parte del proceso de evaluación del software.

Ejemplo:



Imaginemos que en una cocina de restaurante los cocineros tardan mucho porque no tienen los ingredientes ordenados.

Evaluar: sería observar cómo trabajan y notar que pierden tiempo buscando cosas.

Mejorar: sería organizar los ingredientes y herramientas para que cocinen más rápido y con menos errores.

¿Por qué es importante la evaluación de software?

- 1.Garantía de calidad
- 2.Satisfacción del usuario
- 3.Relación costo-eficacia
- 4.Mejora continua

ÉTICA Y PROFESIONALISMO EN LA INGENIERÍA DEL SOFTWARE

Es la responsabilidad que tienen los ingenieros y desarrolladores de software bajo la óptica del cumplimiento y la ética, desde los procesos de levantamiento de información y necesidades para un diseño del sistema, el análisis de resultados, hasta las preguntas críticas sobre el propósito y uso del software.

Ética: actuar correctamente, proteger la privacidad de los datos, no engañar al cliente ni al usuario.





En Costa Rica, la ley establece que alterar un sistema informático puede ser penalizado con dos a ocho años de prisión, según lo dispuesto en el Artículo 231 del Código Penal, cuando la conducta se realiza mediante manipulación informática, programas maliciosos o el uso de tecnologías de la información y la comunicación, y se busca dañar la información o el sistema mismo.

Detalles de la ley:

Artículo 231 (Espionaje Informático y similares):
Se impondrá prisión de dos a ocho años a quien, sin autorización, se apodere, transmita, copie, modifique, destruya, utilice, bloquee o reenvíe información con fines ilícitos.

Contexto del delito:

La alteración de un sistema informático podría encajar en este tipo de delito si la acción implica manipulación de datos o sistemas.

Otras consideraciones:

Fraude Informático (Artículo 217 bis):
Si la alteración se hace con la intención de obtener un beneficio patrimonial, la pena puede variar. Si se manipula información para obtener dinero o bienes, la pena puede ir de tres a seis años, y de cinco a diez años si es contra sistemas públicos o bancarios.

Gravedad y contexto:

La pena específica dependerá del propósito de la alteración (lucro, sabotaje, etc.), del tipo de sistema afectado (público, bancario), y de quién es el autor del delito.

LABORATORIO GIT y GITHUB



Herramienta de control de versiones distribuida

Herramienta de código abierto que los desarrolladores instalan localmente para gestionar el código fuente



Plataforma basada en la nube

Servicio en línea al que los desarrolladores que utilizan Git pueden conectarse y cargar o descargar recursos

ii Feliz noche !!