

# **Capítulo 1:**

# **Desarrollo de**

# **software**

Yoshi Rosanes  
Entornos de Desarrollo  
IES Clara del Rey





# Desarrollo de software

## Contenidos

1. Introducción
2. El software del ordenador
3. Ciclo de vida del software
4. Fases del desarrollo de una aplicación
5. Concepto de programa
6. Lenguajes de programación
7. Obtención de código ejecutable
8. Máquinas virtuales
9. Herramientas utilizadas en programación
10. Ejemplos

# Desarrollo de software



## Objetivos

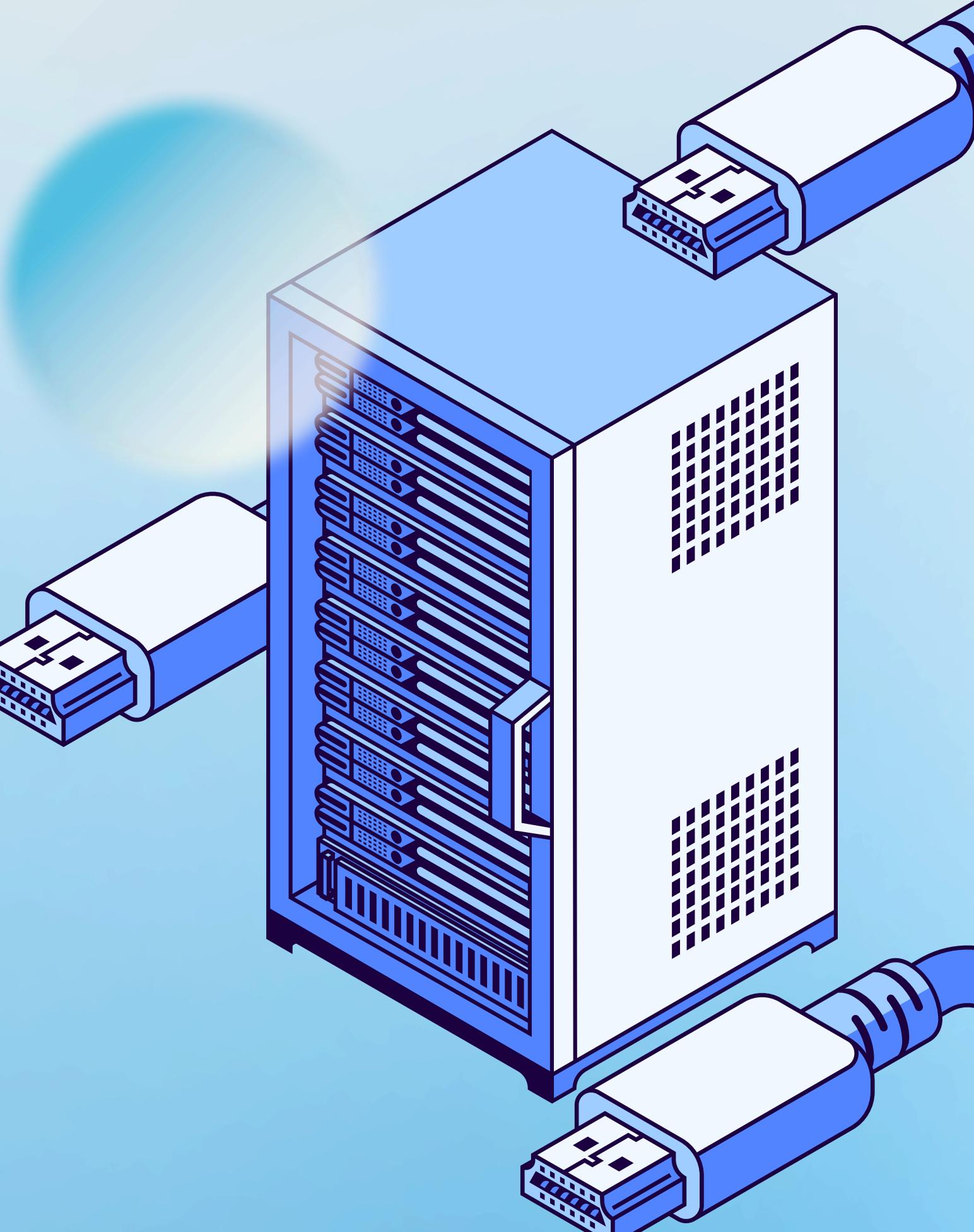
- Reconocer la relación de los programas con los componentes del sistema informático.
- Identificar las fases de desarrollo de una aplicación informática.
- Diferenciar los conceptos de código fuente, código objeto y código ejecutable.
- Clasificar los lenguajes de programación.

# 1. Introducción

El ordenador se compone de dos partes:

- hardware
- software

En este capítulo se estudiarán conceptos básicos que tienen que ver con el desarrollo de software.

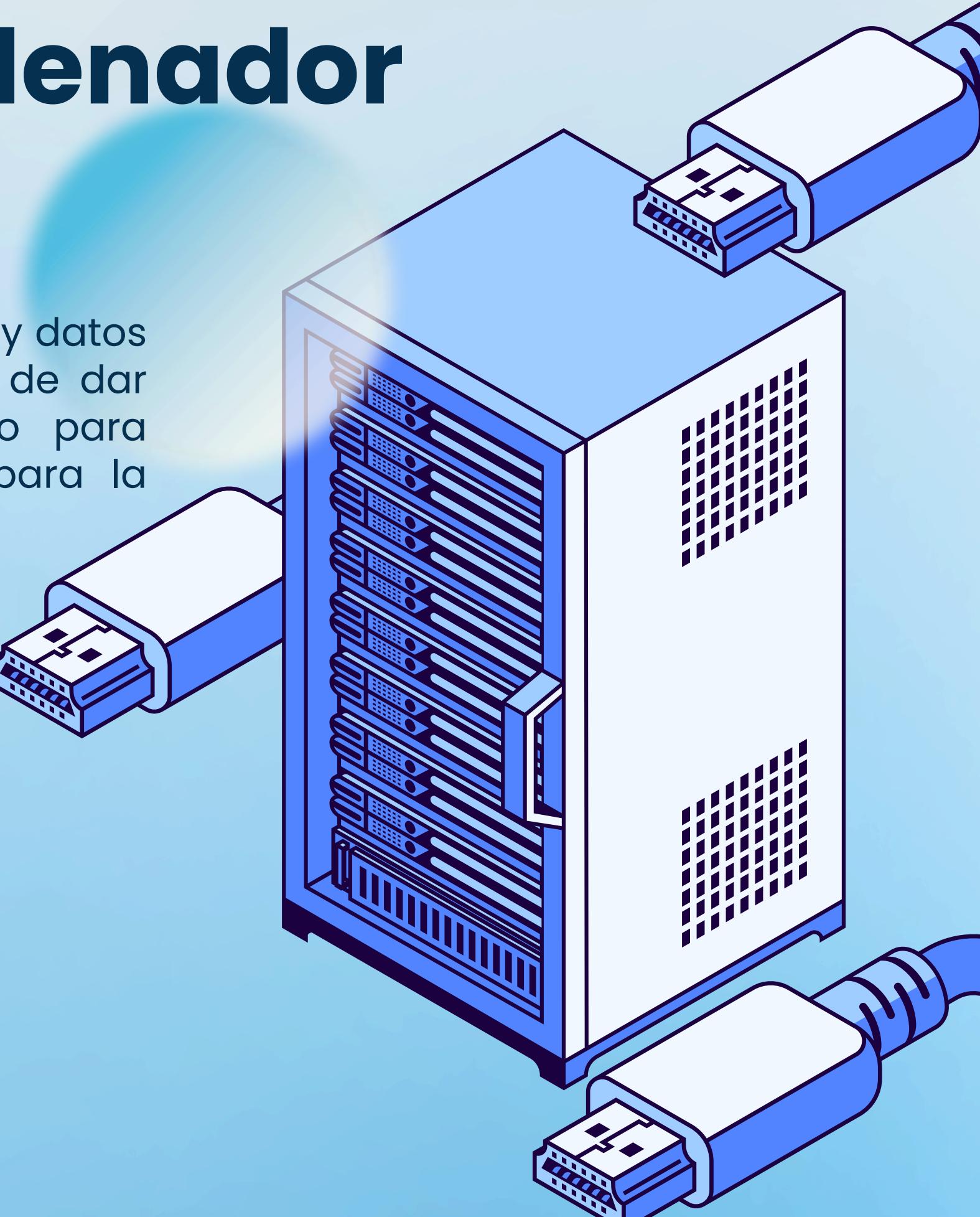


## 2. El software del ordenador

El software es todo aquello que se refiere a los programas y datos almacenados en un ordenador, programas encargados de dar instrucciones para realizar tareas con el hardware o para comunicarnos con otro software, y datos necesarios para la ejecución de los programas.

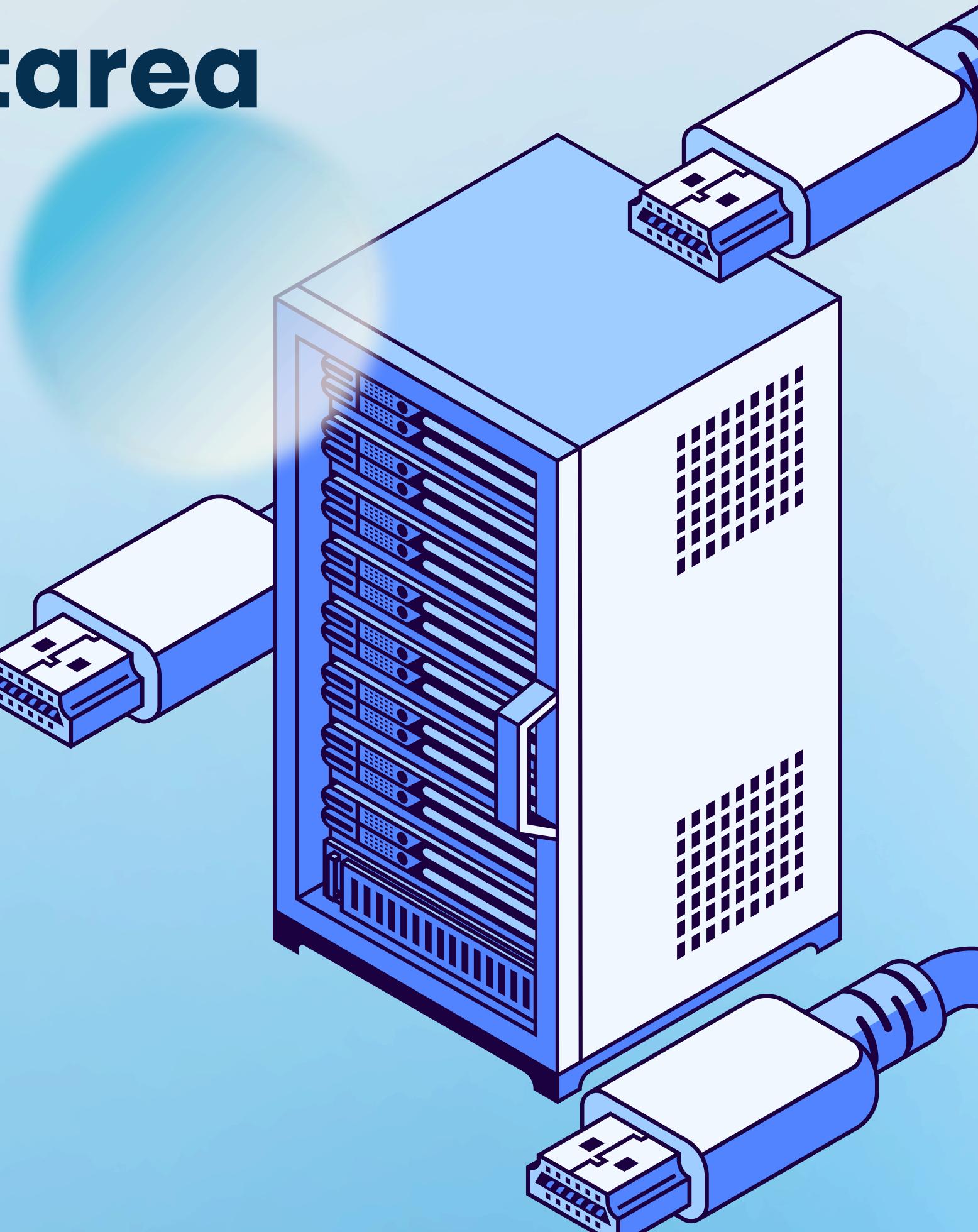
Podemos dividir los tipos de software de 3 maneras diferentes:

- considerando el tipo de tarea que realiza
- el método de distribución
- teniendo en cuenta la licencia.



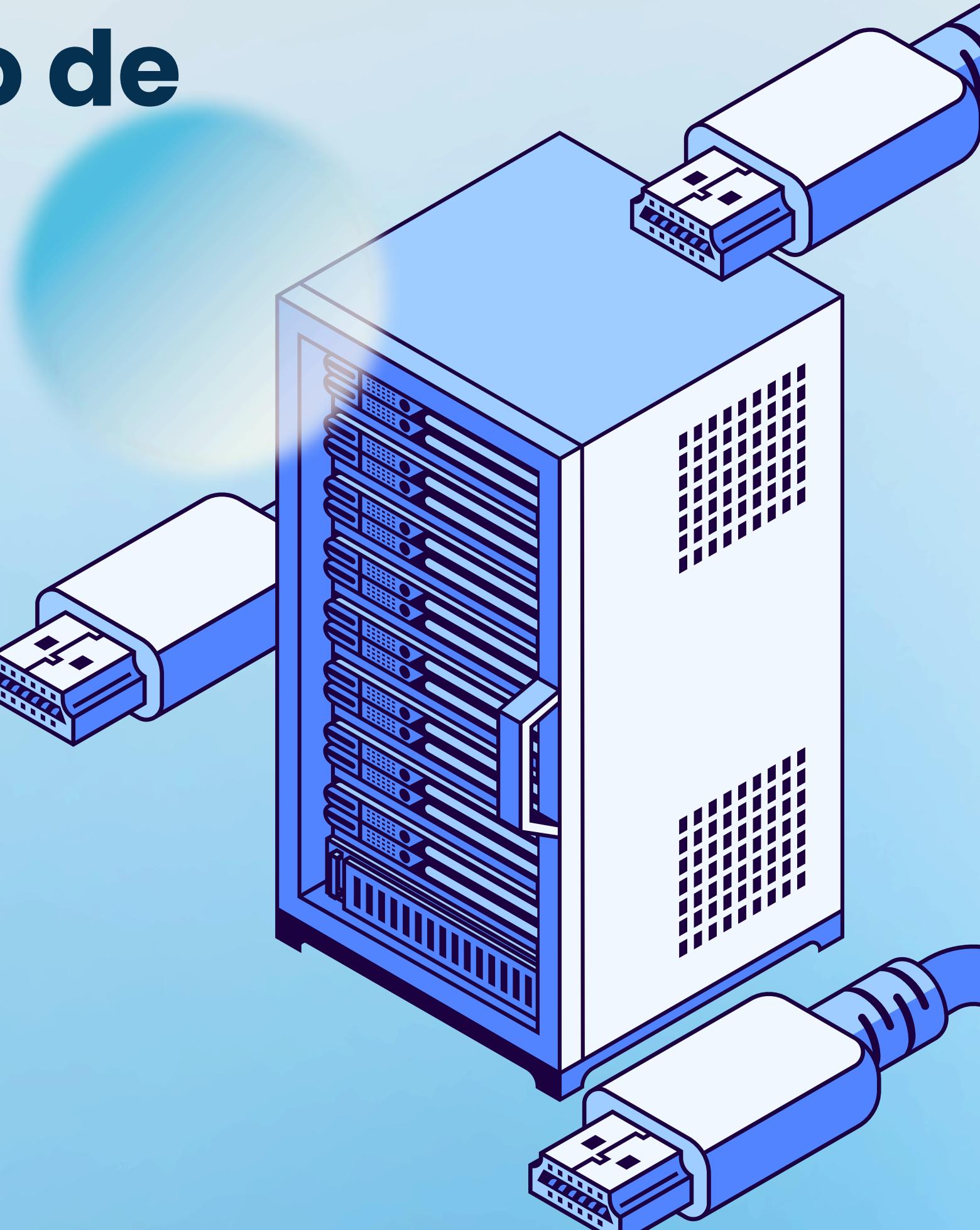
## 2.1. Según el tipo de tarea que realiza

- Software de sistema: es aquel que permite que el hardware funcione.
- Software de aplicación: lo forman programas que nos ayudan a realizar tareas específicas.
- Software de programación o desarrollo: es el que proporciona al programador herramientas para ayudarle a escribir programas informáticos.



## 2.2. Según el método de distribución

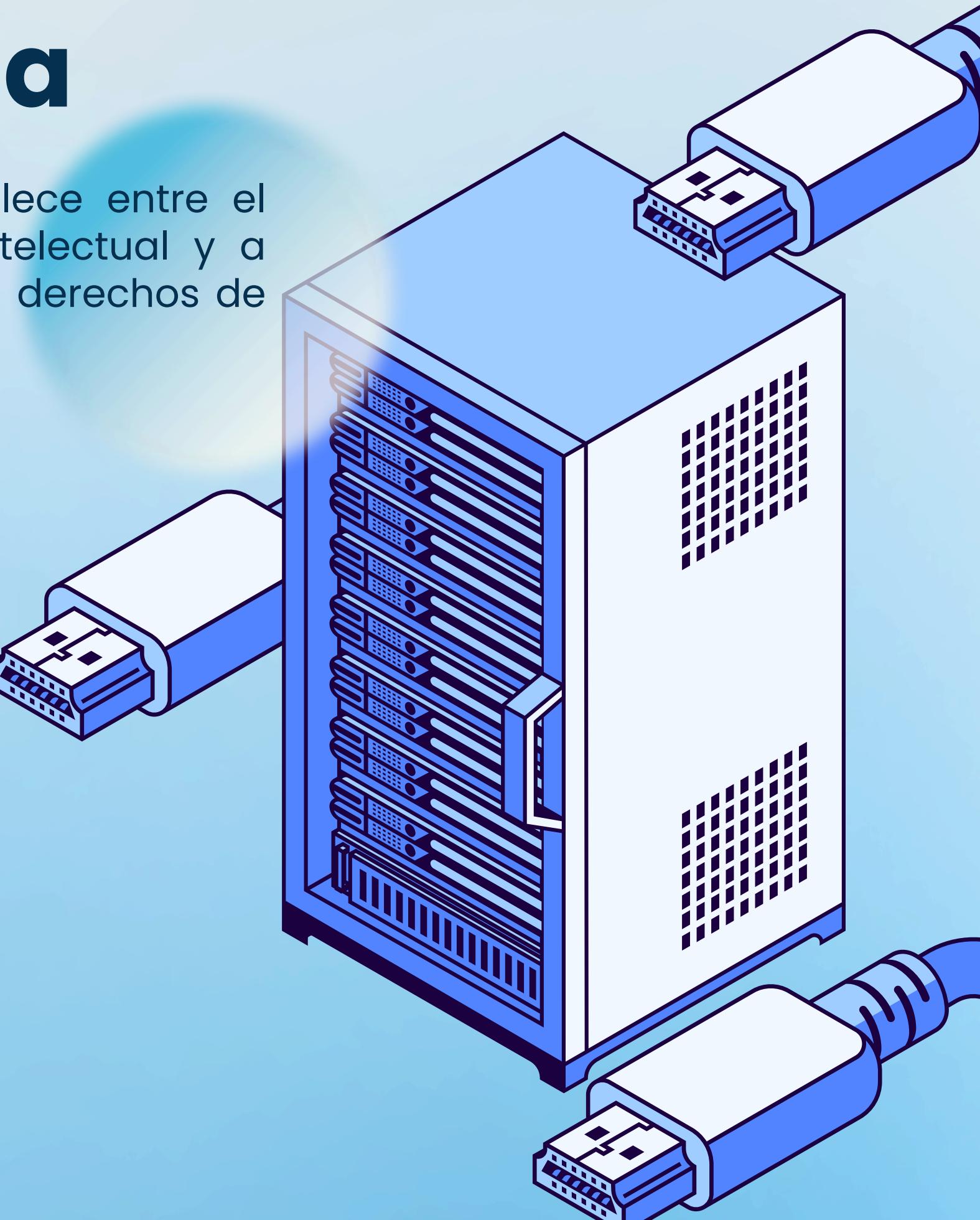
- Shareware: el usuario puede evaluar de forma gratuita el producto por tiempo limitado. Para adquirir el software de manera completa se requiere un pago.
- Freeware: software que se distribuye sin cargo.
- Adware: programas que de forma automática descargan publicidad en nuestro ordenador cuando lo ejecutamos o lo instalamos. Al comprar la licencia se elimina la publicidad.
- Software de uso específico. Requieren de un experto en informática y su precio es elevado.



## 2.3. Según su licencia

Una licencia de software es un contrato que se establece entre el desarrollador de un software sometido a propiedad intelectual y a derechos de autor, y el usuario, en el cual se definen los derechos de ambas partes. Según su licencia, un software puede ser:

- libre: el autor cede una serie de libertades básicas al usuario, como por ejemplo la libertad de distribuir copias a otros usuarios, mejorar o utilizar el programa con cualquier fin. La **licencia GPL** es la más utilizada en software libre. Da derecho a usar y modificar el programa, con la obligación de hacer públicas las versiones modificadas de éste.
- propietario: se distribuye sin acceso al código fuente. El propietario normalmente prohíbe la redistribución, copia o modificación del programa.
- de dominio público: aquel que carece de licencia o no hay forma de determinarla porque se desconoce el autor.



# 3. Ciclo de vida del software

## 3.1. Definición

El ciclo de vida del software es un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo la explotación y el mantenimiento de un software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso.

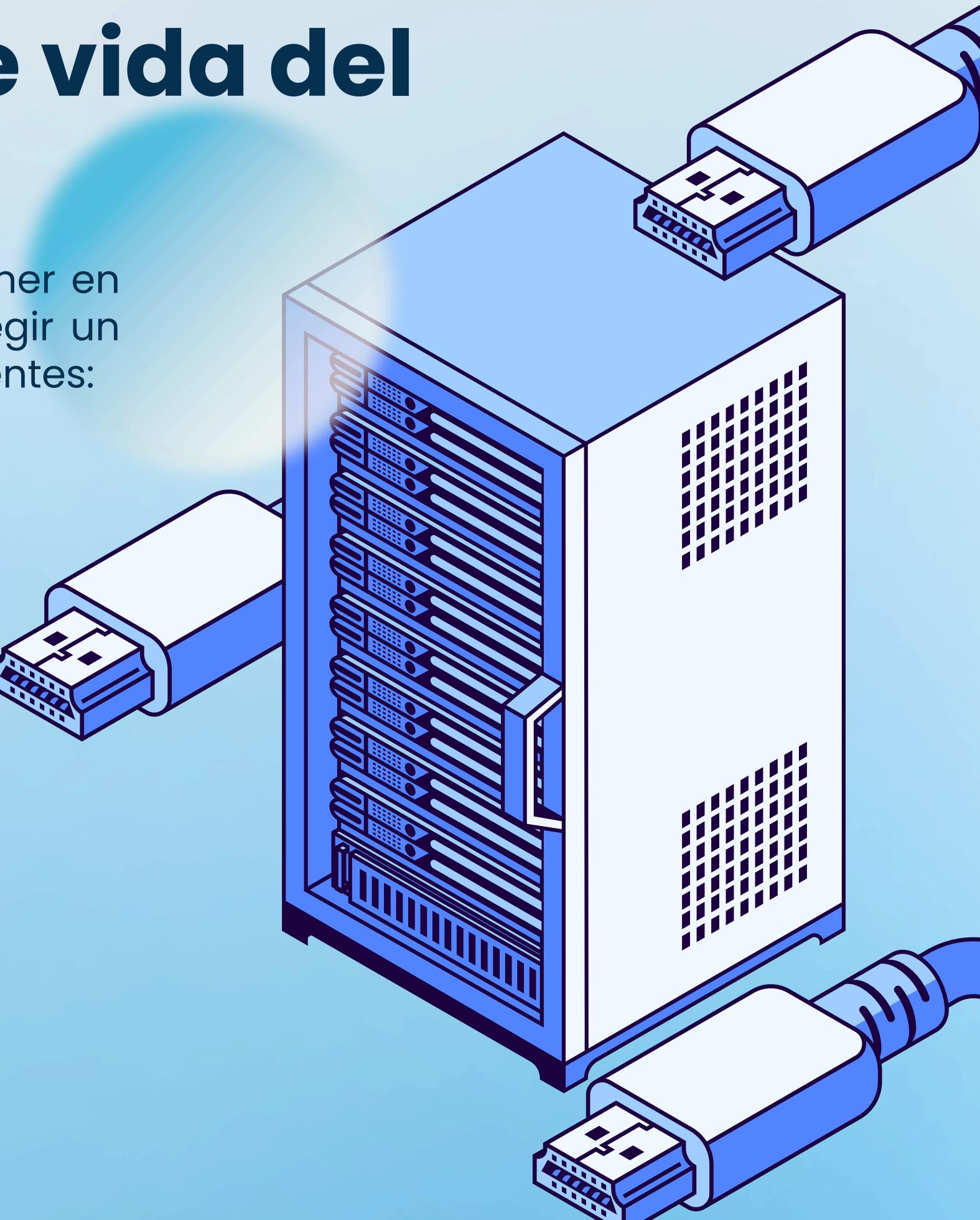
1. **Análisis:** construye un modelo de los requisitos.
2. **Diseño:** en esta etapa ya sabemos qué es lo que hay que hacer, ahora hay que definir cómo se va a resolver el problema.
3. **Codificación:** en esta etapa se traduce lo descrito en el diseño a un código ejecutable.
4. **Pruebas:** se comprueba que se cumplen criterios de corrección y calidad.
5. **Mantenimiento:** esta fase tiene lugar después de la entrega de software al cliente. Se producen cambios porque se han encontrado errores, porque es necesario adaptarse al entorno o porque el cliente requiere mejoras funcionales.



## 3.2. Modelos de ciclo de vida del software

Existen varios modelos de ciclo de vida, es importante tener en cuenta las características del proyecto software para elegir un modelo u otro. Los modelos más importantes son los siguientes:

- Cascada.
- Incremental.
- Espiral.



## 3.2.1. Ciclo de vida en cascada

En este modelo las etapas para el desarrollo de software tienen un orden, de tal forma que para empezar una etapa es necesario finalizar la etapa anterior.

Después de cada etapa se realiza una revisión para comprobar si se puede pasar a la siguiente.

Si se tiene que volver a una de las etapas anteriores hay que recorrer de nuevo el resto de etapas.

Se recomienda cuando:

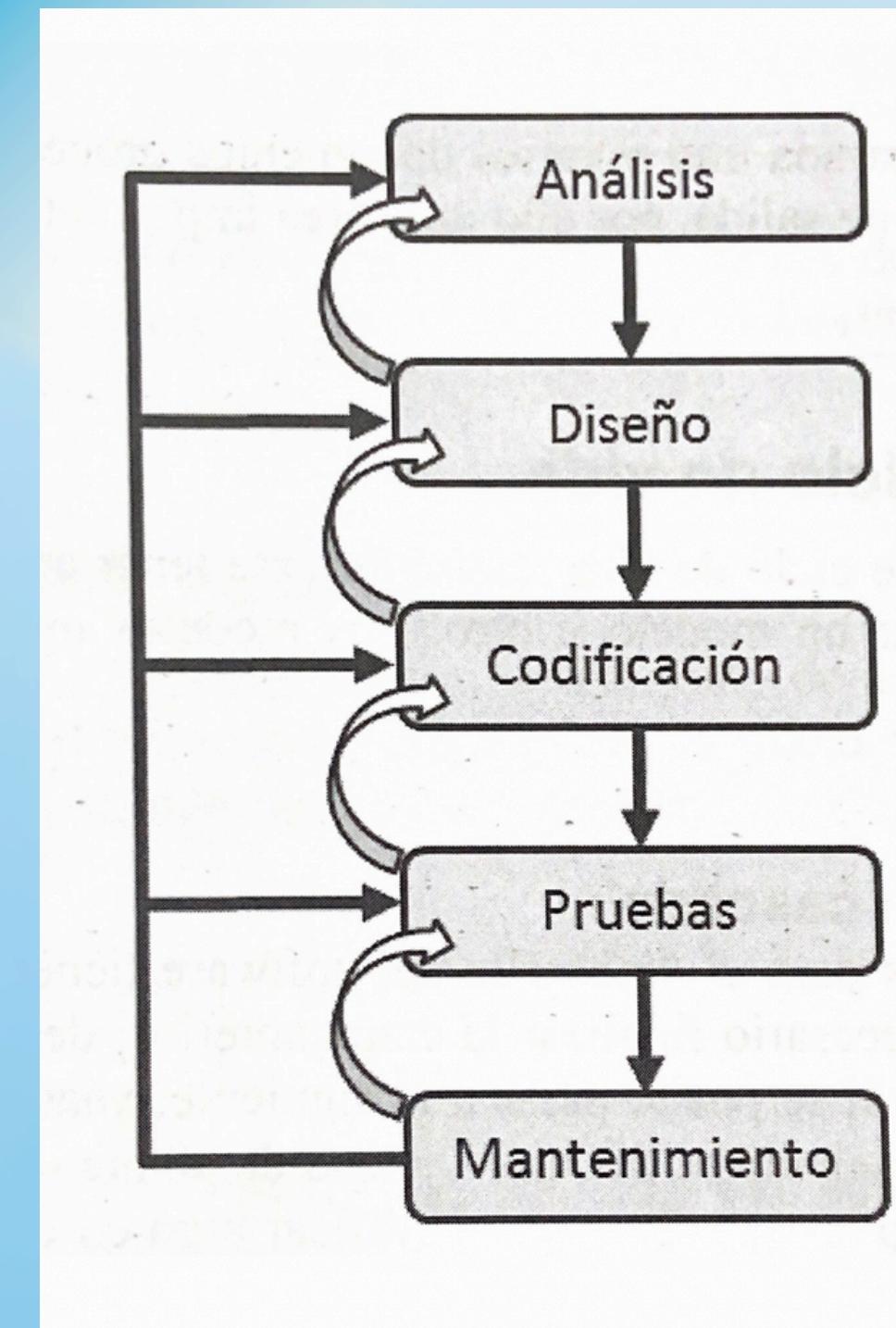
- el proyecto es similar a alguno que ya se haya realizado con éxito anteriormente.
- Los requisitos son estables y están bien comprendidos.
- Los clientes no necesitan versiones intermedias.



## 3.2.1. Ciclo de vida en cascada

Tiene varias variantes, una de las más utilizadas es la que produce una realimentación entre etapas, se la conoce como **Modelo en Cascada con Realimentación**.

Supongamos que la etapa de Análisis ha finalizado y se puede pasar a la de Diseño. Durante el desarrollo de esta etapa se detectan fallos, entonces será necesario retornar a la etapa anterior, realizar los ajustes pertinentes y continuar de nuevo con el Diseño. A esto se le conoce como realimentación, pudiendo volver de una etapa a la anterior o incluso varias etapas a la anterior.



## 3.2.2. Modelos evolutivos

La competencia en el mercado del software es tan grande que las empresas no pueden esperar a tener un producto totalmente completo para lanzarlo al mercado.

Los dos modelos que quedan por estudiar son modelos evolutivos, que permiten desarrollar versiones cada vez más completas hasta llegar al producto final deseado.

En estos modelos se asume que las necesidades del usuario no están completas y se requiere una vuelta a planificar y diseñar después de cada implantación de los entregables.

Los modelos evolutivos más conocidos son: el **Iterativo incremental** y el **Espiral**.



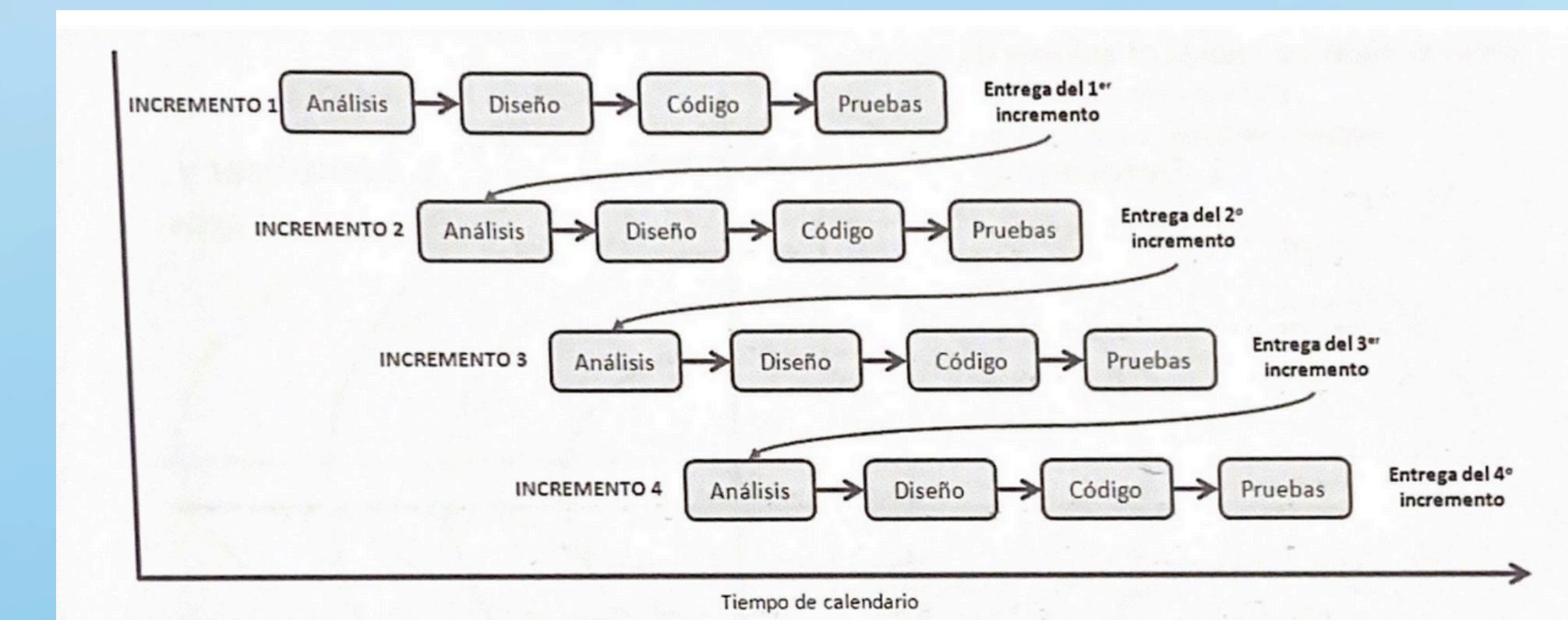
## 3.2.2. Modelos evolutivos

### Modelo Iterativo incremental

Está basado en varios ciclos cascada realimentados aplicados repetidamente. El modelo incremental entrega el software en partes pequeñas, pero utilizables, llamadas **incrementos**. En general, cada incremento se construye sobre aquel que ya ha sido entregado.

Se recomienda cuando:

- Los requisitos o el diseño no están completamente definidos y es posible que haya grandes cambios.
- Se están probando o introduciendo nuevas tecnologías.



## 3.2.2. Modelos evolutivos

### Modelo en Espiral

Este modelo combina el Modelo en Cascada con el modelo iterativo. El proceso se representa como una espiral, donde en cada ciclo se desarrolla una parte del mismo. Cada ciclo está formado por cuatro fases, como se ve en la siguiente figura y cuando termina produce una versión incremental del software con respecto al ciclo anterior. En este aspecto se parece al Modelo Iterativo.

Se recomienda cuando:

- Proyectos de gran tamaño y que necesitan constantes cambios.
- Proyectos donde sea importante el factor riesgo.



# 4. Fases del desarrollo de una aplicación

Antes de desarrollar un proyecto hay que elegir un modelo de ciclo de vida. En la sección anterior hemos visto las diferentes etapas del ciclo de vida de un software. En esta sección se desarrollan estas etapas.



# 4. Fases del desarrollo de una aplicación

## 4.1. Análisis

Lo más importante del éxito de un proyecto de software es entender y comprender el problema que se necesita resolver.

En esta fase se analizan y especifican los **requisitos** del sistema.

Para facilitar la comunicación entre el cliente y los desarrolladores se utilizan varias técnicas:

- Entrevistas
- Desarrollo conjunto de aplicaciones
- Planificación conjunta de requisitos
- Brainstorming
- Prototipos
- Casos de uso

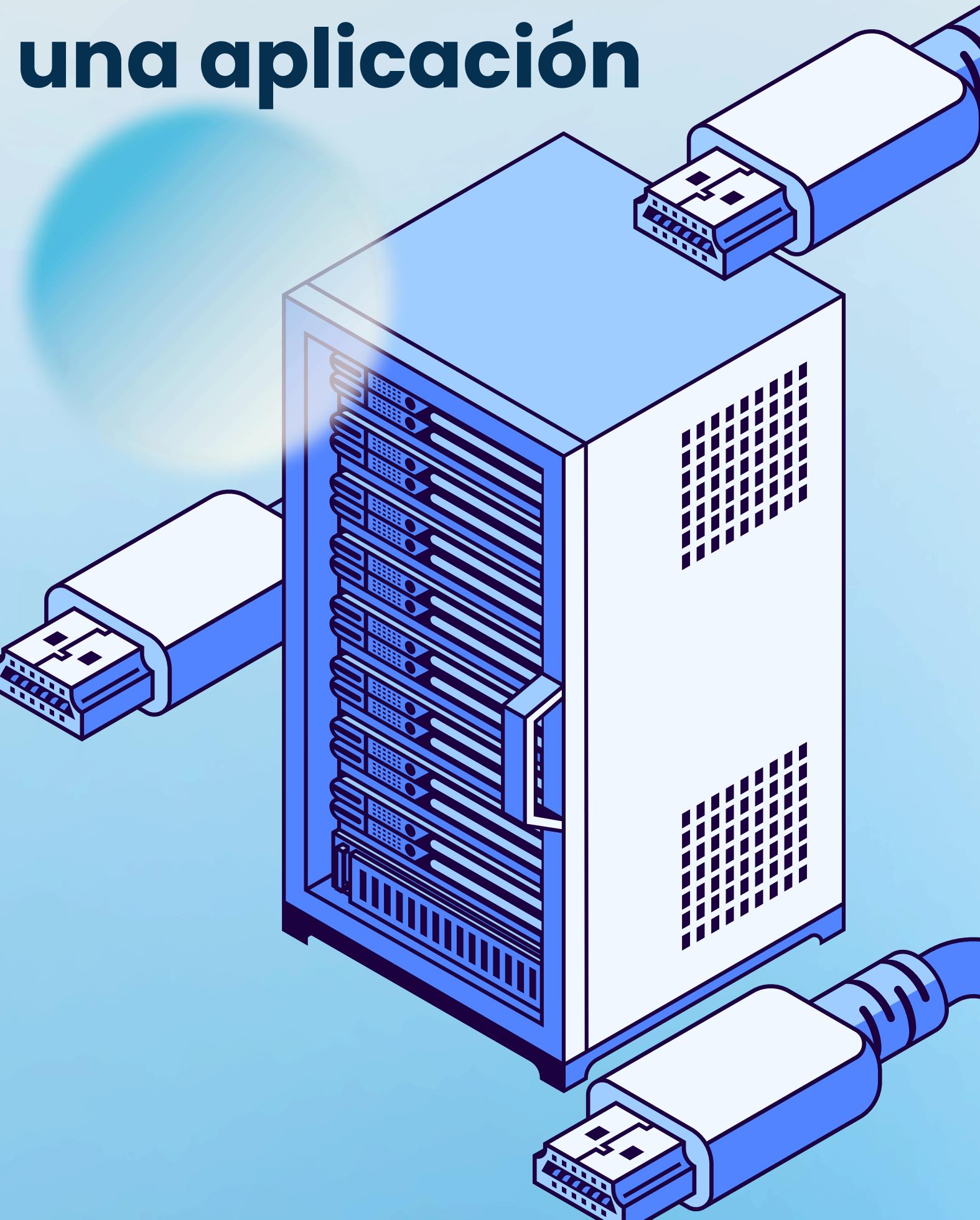


# 4. Fases del desarrollo de una aplicación

## 4.1. Análisis

Se especifican dos tipos de requisitos:

- Funcionales: describen con detalle la función que realiza el sistema, cómo reacciona antes determinadas entradas, cómo se comporta en situaciones particulares, etc.
- No funcionales: tratan sobre las características del sistema, como puede ser la fiabilidad, mantenibilidad, sistema operativo, etc.



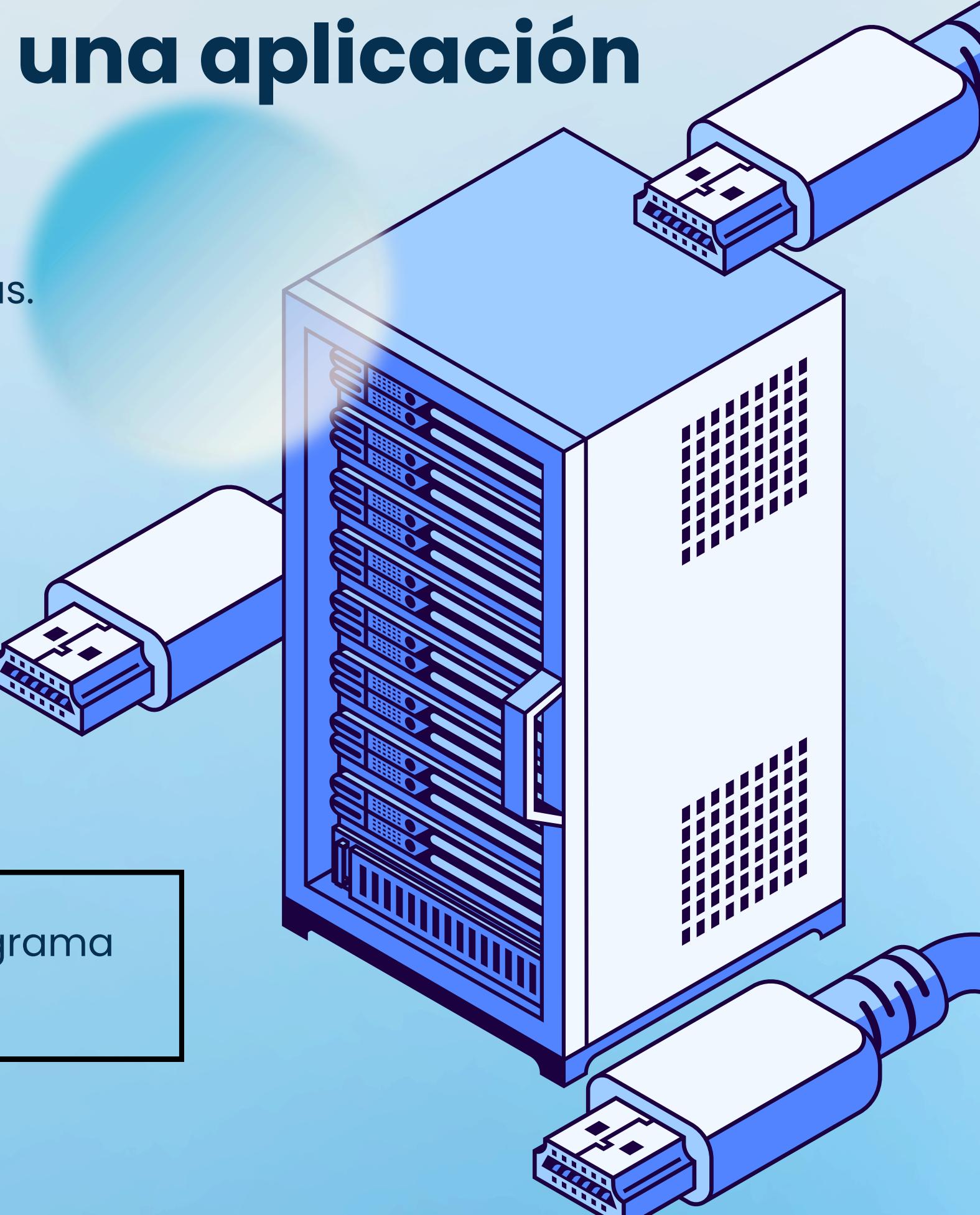
# 4. Fases del desarrollo de una aplicación

## 4.1. Análisis

Para representar los requisitos se utilizan diferentes técnicas.

- Diagramas de flujo de datos, DFD.
- Diagramas de flujo de control, DFC.
- Diagramas de transición de estados, DTE.
- Diagrama Entidad/Relación, DER.
- Diccionario de datos, DD.

Investiga sobre estas técnicas e intenta desarrollar un diagrama sencillo en cada caso. Puedes utilizar draw.io



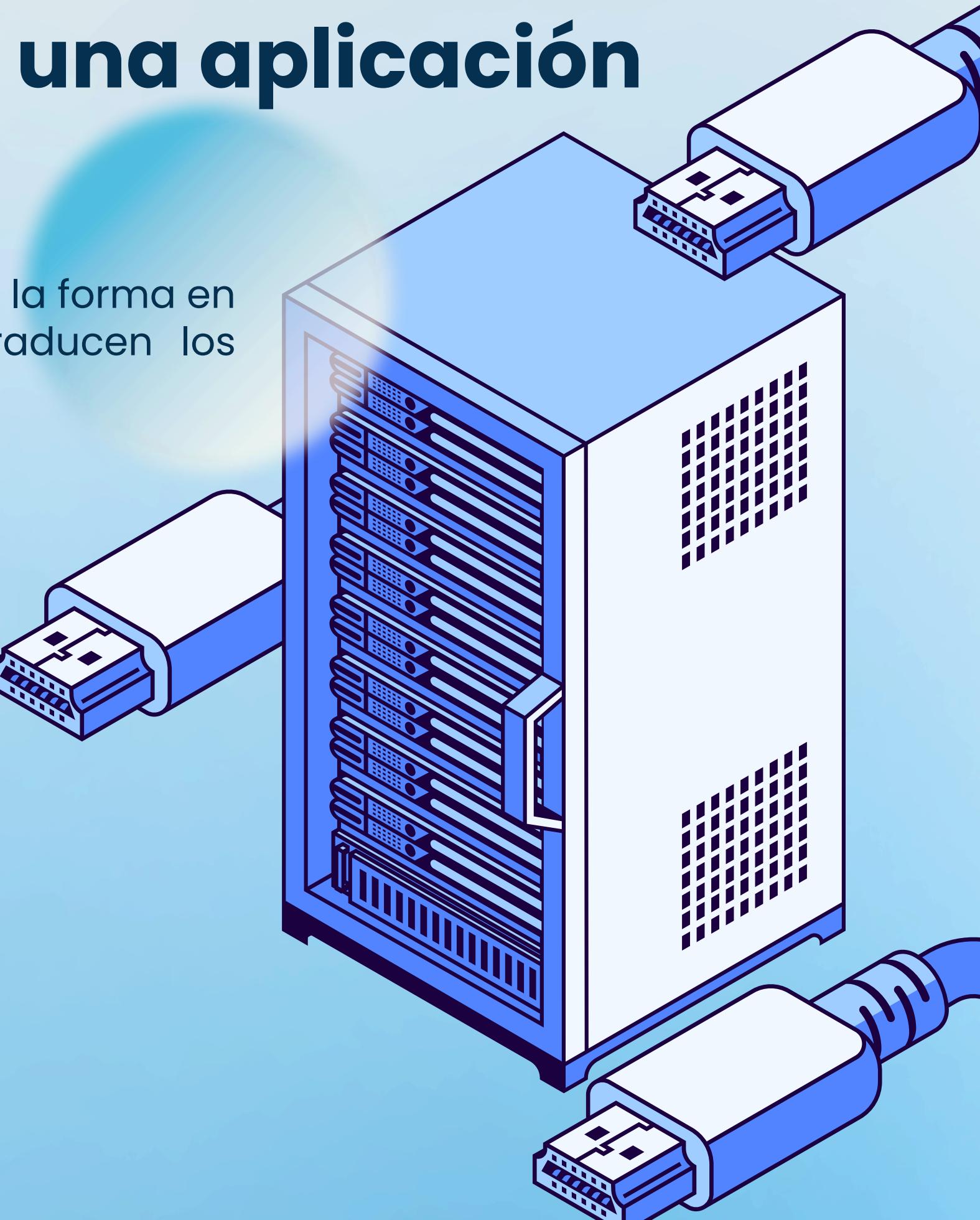
# 4. Fases del desarrollo de una aplicación

## 4.2. Diseño

Una vez identificados los requisitos es necesario componer la forma en que se solucionará el problema. En esta etapa se traducen los requisitos funcionales en una representación de software.

Principalmente hay dos tipos de diseño:

- diseño estructurado: basado en el flujo de datos a través del sistema.
- diseño orientado a objetos: donde el sistema se entiende como un conjunto de objetos que tienen propiedades y comportamientos, y eventos que activan operaciones que modifican el estado de los objetos.



# 4. Fases del desarrollo de una aplicación

## 4.2. Diseño

### 4.2.1. Diseño estructurado: construcciones lógicas

El diseño estructurado tiene 3 construcciones lógicas principales:

- La construcción secuencial.
- La condicional.
- El bucle.

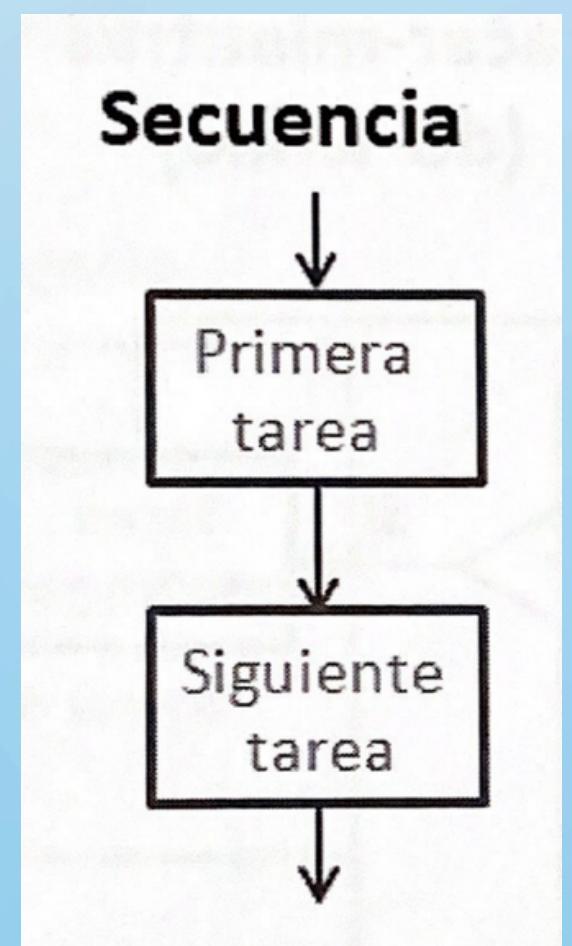


# 4. Fases del desarrollo de una aplicación

## 4.2. Diseño

### 4.2.1. Diseño estructurado: construcciones lógicas

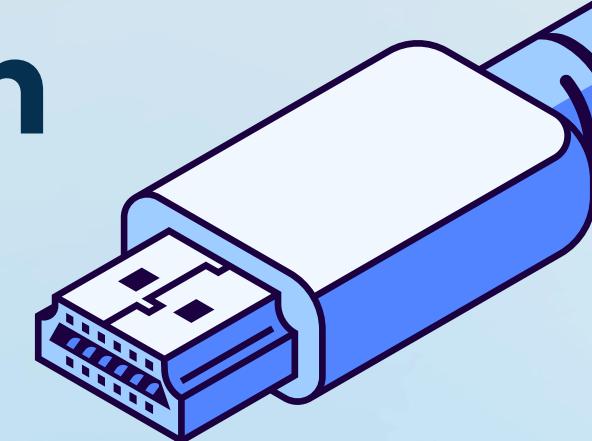
La **construcción secuencial** implementa los pasos del proceso esenciales para la especificación de cualquier algoritmo.



# 4. Fases del desarrollo de una aplicación

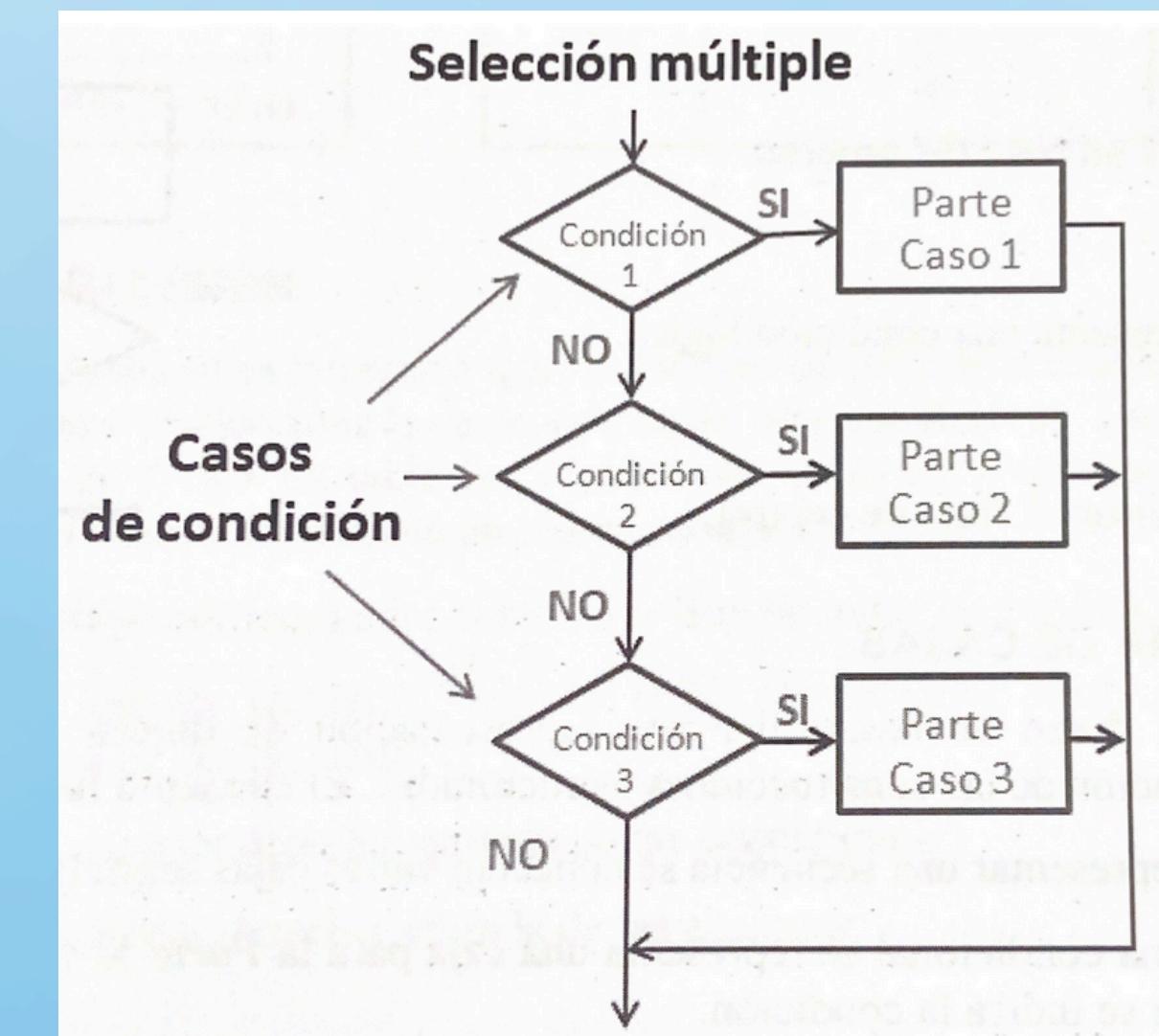
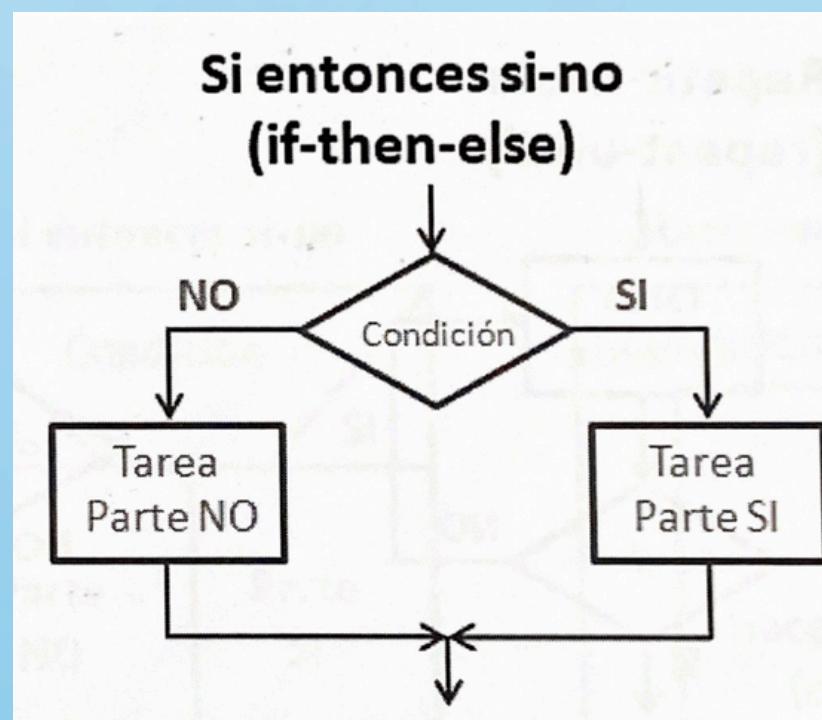
## 4.2. Diseño

### 4.2.1. Diseño estructurado: construcciones lógicas



La **construcción condicional** permite seleccionar un proceso u otro a partir de la evaluación de una condición lógica que se representa mediante un rombo. Si se cumple la condición se realiza la tarea de la Parte SI si no se cumple se realiza la tarea de la Parte NO.

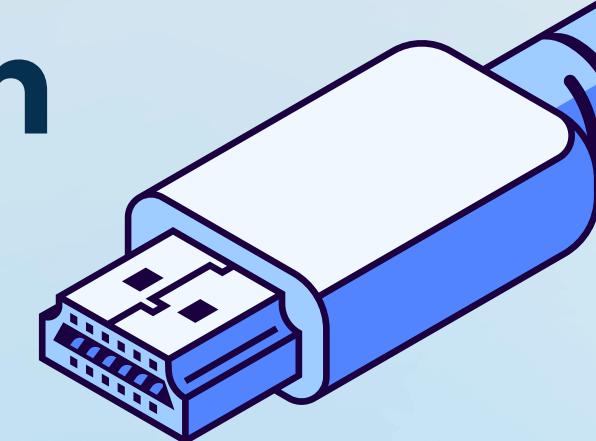
La **selección múltiple** es una extensión de esta estructura.



# 4. Fases del desarrollo de una aplicación

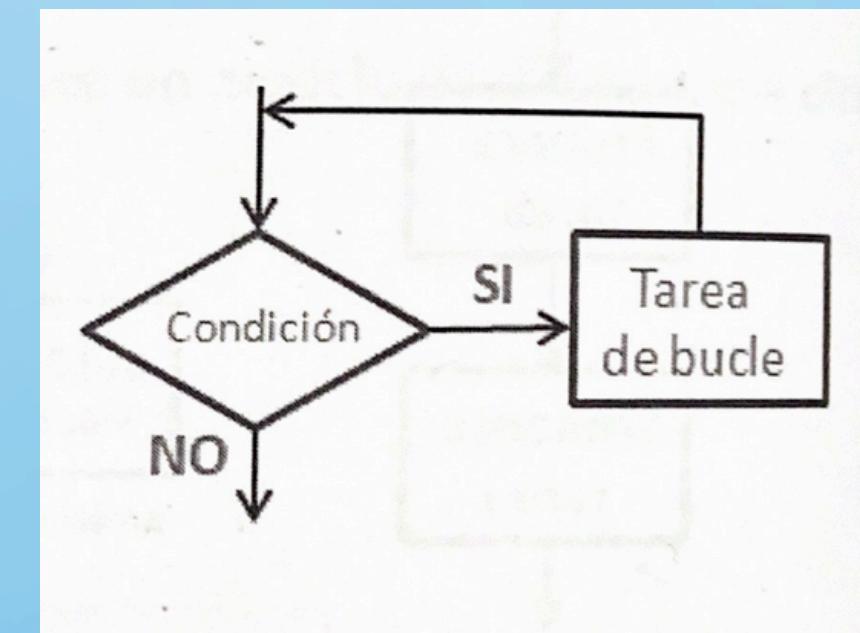
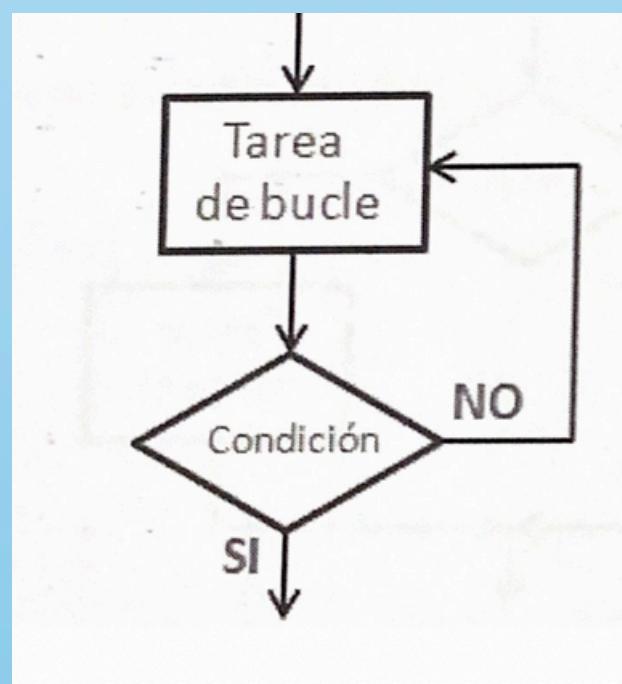
## 4.2. Diseño

### 4.2.1. Diseño estructurado: construcciones lógicas



En la **construcción bucle**, distinguimos dos tipos:

- **Do while**: primero ejecuta la tarea del bucle y después comprueba la condición, si no se cumple se vuelve a realizar la tarea; si la condición se cumple finaliza el bucle. La tarea se realiza al menos una vez.
- **While**: primero se comprueba la condición y después se realiza la tarea del bucle repetidamente siempre y cuando la condición se cumpla; el bucle finaliza cuando la condición no se cumple.



# 4. Fases del desarrollo de una aplicación

## 4.2. Diseño

### 4.2.1. Diseño estructurado: herramientas gráficas

Para representar el diseño se emplean algunas **herramientas gráficas**. Vamos a ver en detalle 4 herramientas diferentes:

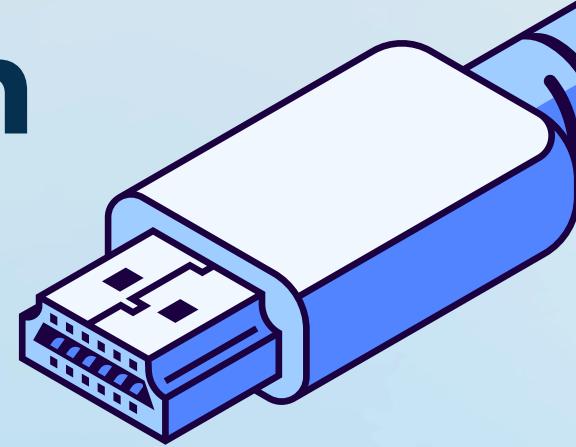
- Diagrama de flujo
- Diagrama de cajas
- Tablas de decisión
- Pseudocódigo



# 4. Fases del desarrollo de una aplicación

## 4.2. Diseño

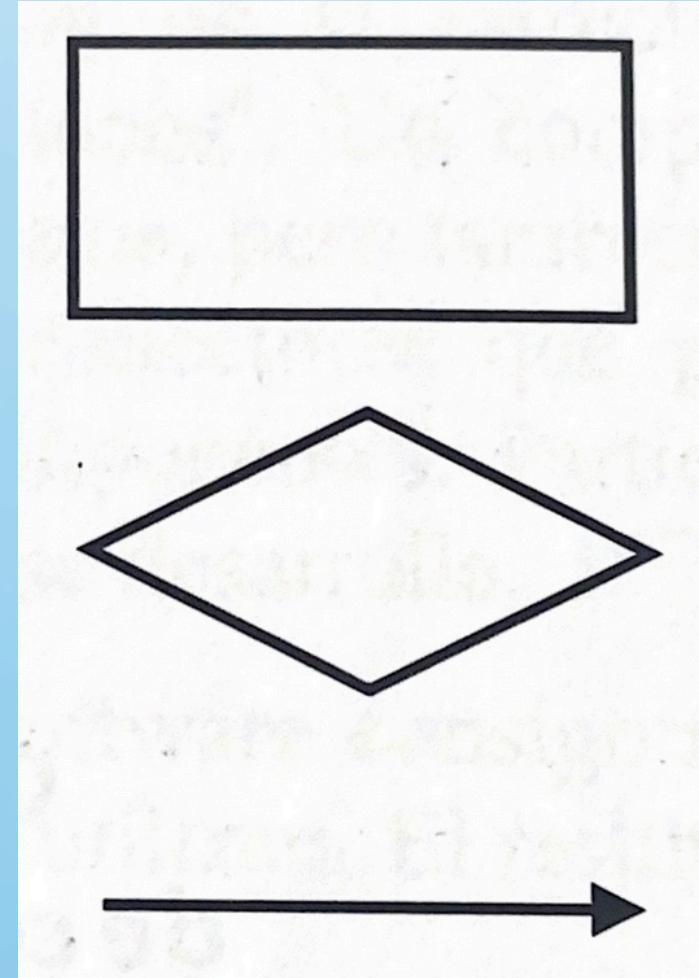
### 4.2.1. Diseño estructurado: herramientas gráficas



El **diagrama de flujo** es una herramienta muy usada para el diseño procedural.

Se utilizan los símbolos vistos anteriormente:

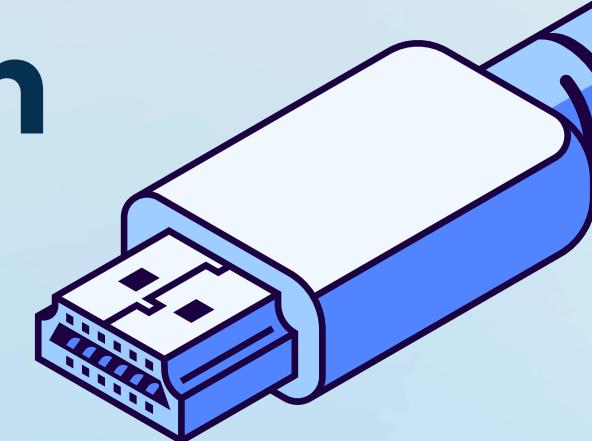
- Una caja indica un paso del proceso.
- Un rombo representa una condición lógica.
- Las flechas indican el flujo de control.



# 4. Fases del desarrollo de una aplicación

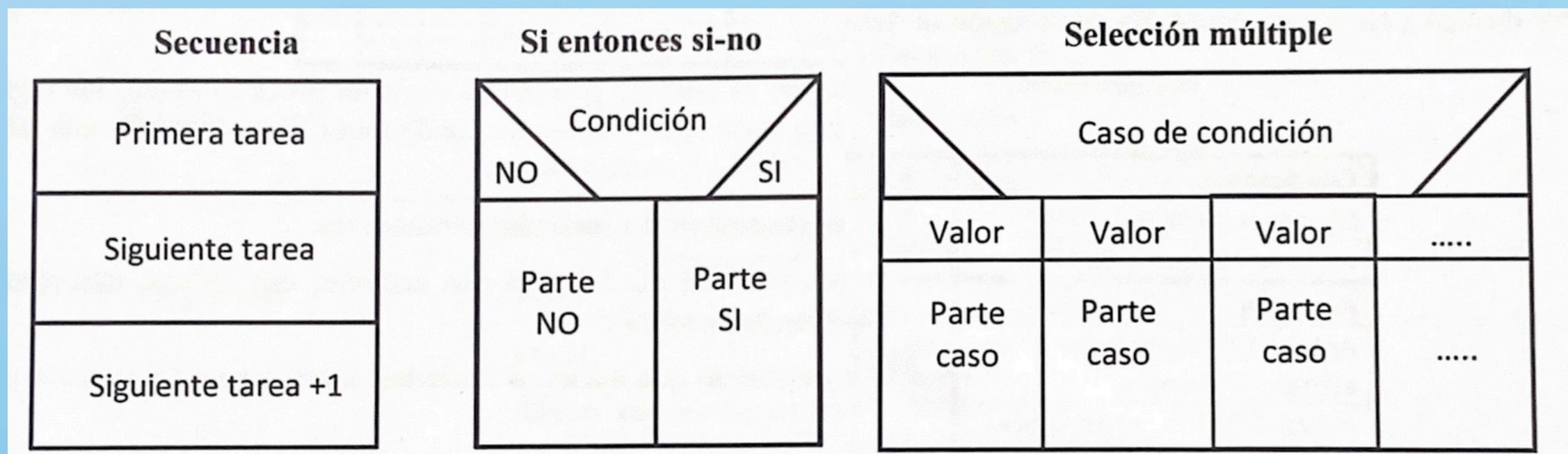
## 4.2. Diseño

### 4.2.1. Diseño estructurado: herramientas gráficas



El diagrama de cajas:

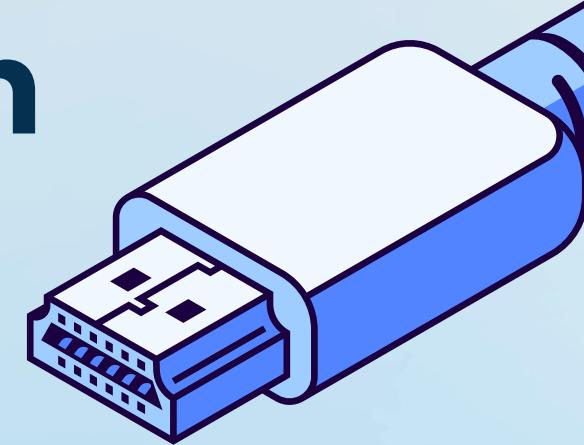
- Para representar una secuencia se conectan varias cajas seguidas.
- Para una condicional se representa una caja para la Parte SI y otra para la Parte NO, encima se indica la condición.
- En la selección múltiple en la parte superior se indica la condición, se definen tantas columnas como valores se vayan a comprobar en la condición, debajo de cada valor se indica la instrucción a realizar.



# 4. Fases del desarrollo de una aplicación

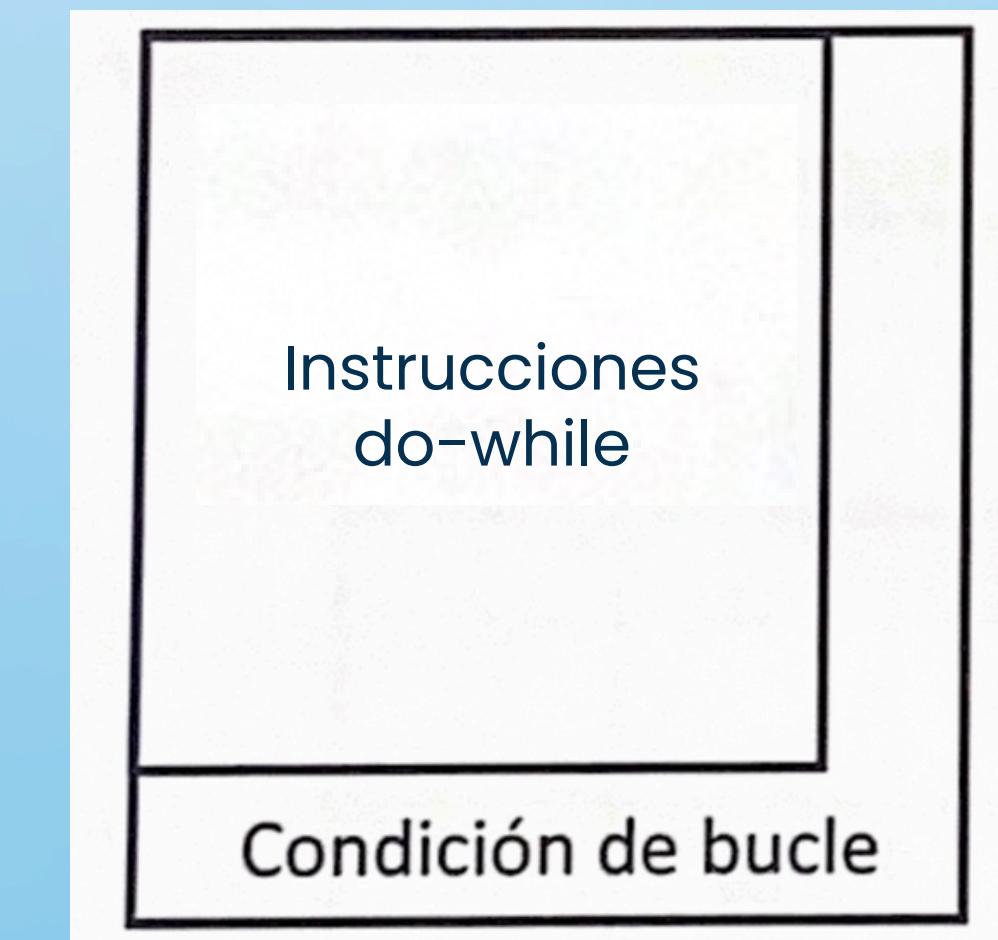
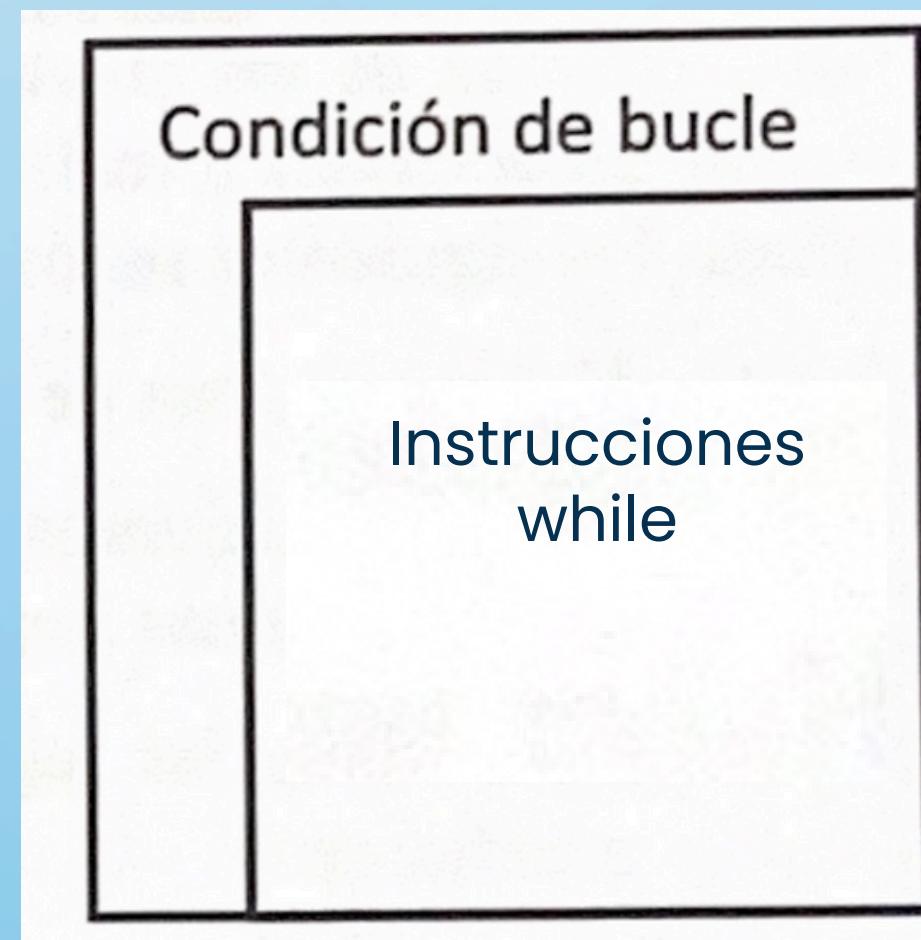
## 4.2. Diseño

### 4.2.1. Diseño estructurado: herramientas gráficas



El **diagrama de cajas**:

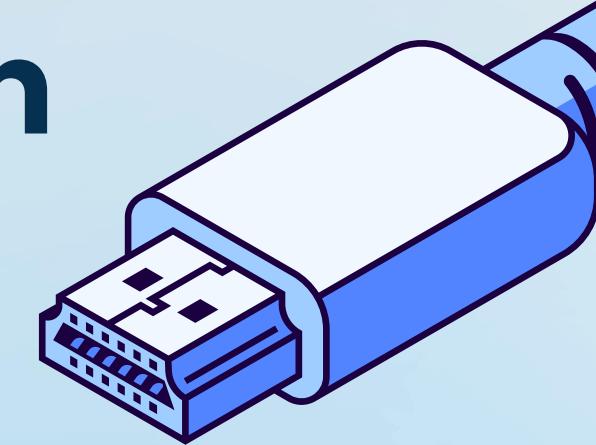
- En los bucles el proceso a repetir se encierra en una caja que está dentro de otra caja donde en la parte superior (while) o inferior (do-while) se indica la condición del bucle.



# 4. Fases del desarrollo de una aplicación

## 4.2. Diseño

### 4.2.1. Diseño estructurado: herramientas gráficas



Para construir una **tabla de decisión** se realizan los siguientes pasos:

- Hacer una lista de todas las acciones y condiciones.
- Asociar conjuntos específicos de condiciones con acciones específicas, eliminando combinaciones imposibles de condiciones.
- Determinar las reglas indicando que acción o acciones ocurren para un conjunto de condiciones.

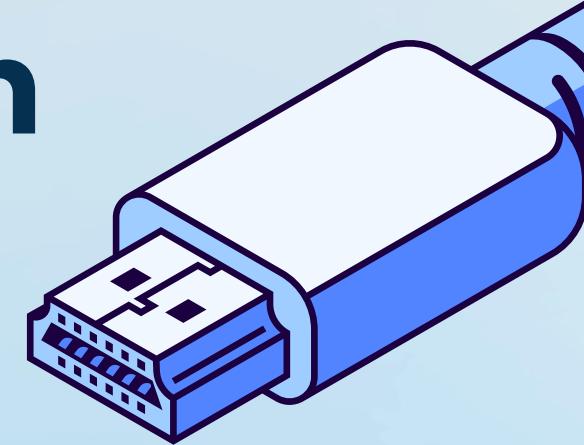
Condiciones	Reglas				
	1	2	3	4	n
Condición nº 1	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>		
Condición nº 2		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
Condición nº 3	<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>	
<b>Acciones</b>					
Acción nº1	<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>	
Acción nº 2	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
Acción nº 3		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Acción nº 4			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Condiciones	Reglas			
	1	2	3	4
Cliente registrado	SI	SI	NO	NO
Importe compra > 800 €	SI	NO	SI	NO
<b>Acciones</b>				
Aplicar 1% Bonificación sobre el importe compra	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
Aplicar 3% Descuento sobre el importe compra	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	
Calcular Factura	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

# 4. Fases del desarrollo de una aplicación

## 4.2. Diseño

### 4.2.1. Diseño estructurado: herramientas gráficas



El **pseudocódigo** utiliza texto descriptivo para realizar el diseño de un algoritmo, mezclando frase en lenguaje natural con estructuras sintácticas que incluyen palabras clave que permiten construir las estructuras básicas de la programación estructurada.

#### **Inicio**

```

    Abrir Fichero
    Leer Registro del Fichero
    Mientras no sea Fin de Fichero Hacer
        Procesar Registro leído
        Leer Registro del Fichero
    Fin mientras
    Cerrar Fichero
Fin.

```

Secuencial	Instrucción 1 Instrucción 2 ..... Instrucción n
Condicional	<b>Si</b> <condición> <b>Entonces</b> <Instrucciones> <b>Si no</b> <Instrucciones> <b>Fin si</b>
Condicional múltiple.	<b>Según</b> sea <variable> <b>Hacer</b> <b>Caso</b> valor 1: <Instrucciones> <b>Caso</b> valor 2: <Instrucciones> <b>Caso</b> valor 3: <Instrucciones> <b>Otro caso:</b> <Instrucciones> <b>Fin según</b>
Repetir-hasta	<b>Repetir</b> <instrucciones> <b>Hasta que</b> <condición >
Hacer-mientras	<b>Mientras</b> <condición> <b>Hacer</b> <instrucciones> <b>Fin mientras</b>

# 4. Fases del desarrollo de una aplicación

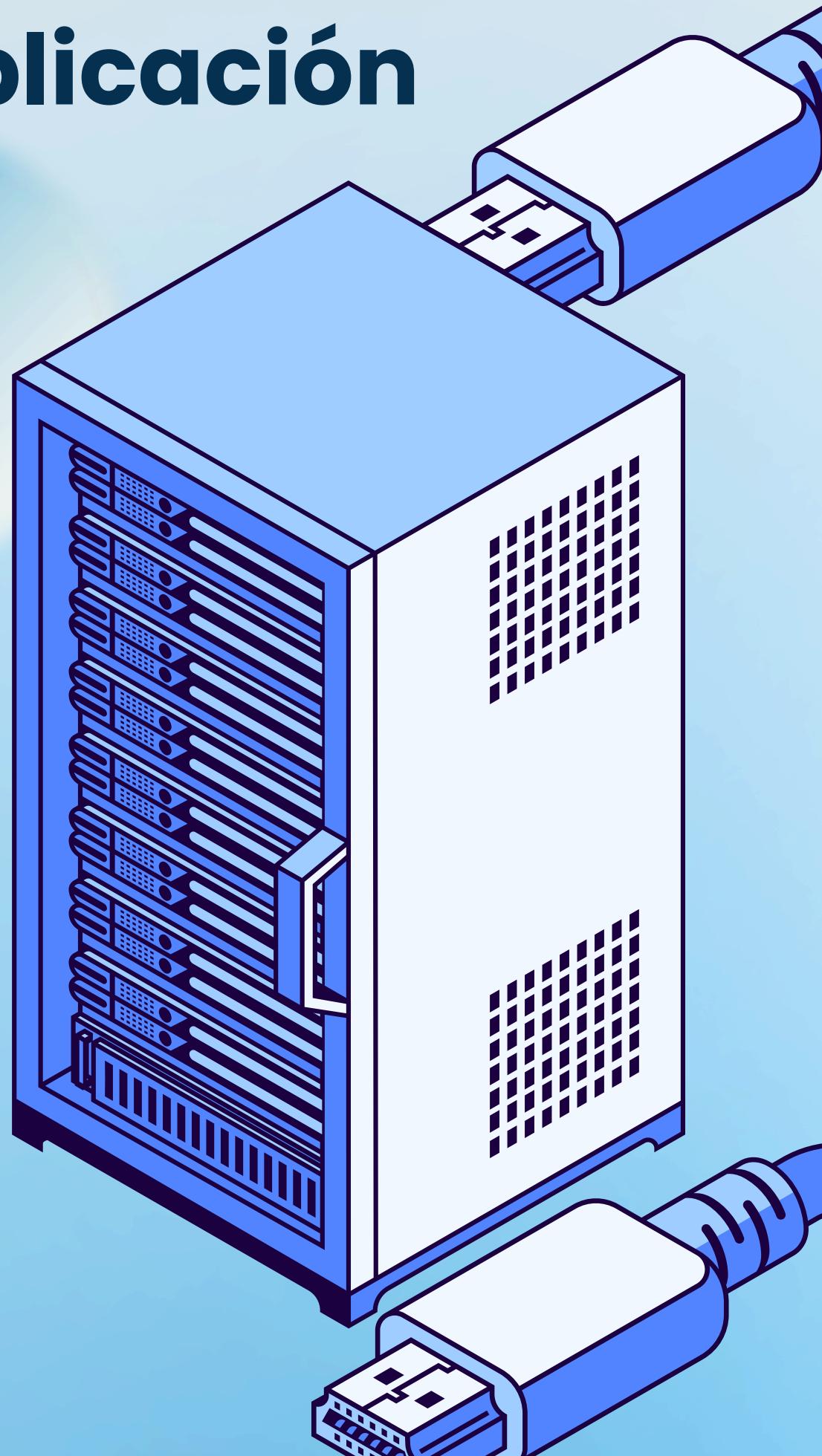
## 4.2. Diseño

### 4.2.2. Diseñado orientado a objetos

Para llevar a cabo el diseño de software orientado a objetos (DOO) hay que partir de un análisis orientado a objetos (AOO). En dicho análisis se definen todas las clases que son importantes para el problema, las operaciones, atributos asociados, relaciones, comportamientos y las comunicaciones entre clases.

El diseño orientado a objetos define 4 capas de diseño:

- **Subsistema:** se centra en el diseño de los subsistemas que implementan las funciones principales del sistema.
- **Clases y objetos:** especifica la arquitectura de objetos global y la jerarquía de clases requerida para implementar un sistema.
- **Mensajes:** indica como se realiza la colaboración entre los objetos.
- **Responsabilidades:** identifica las operaciones y atributos que caracterizan cada clase.



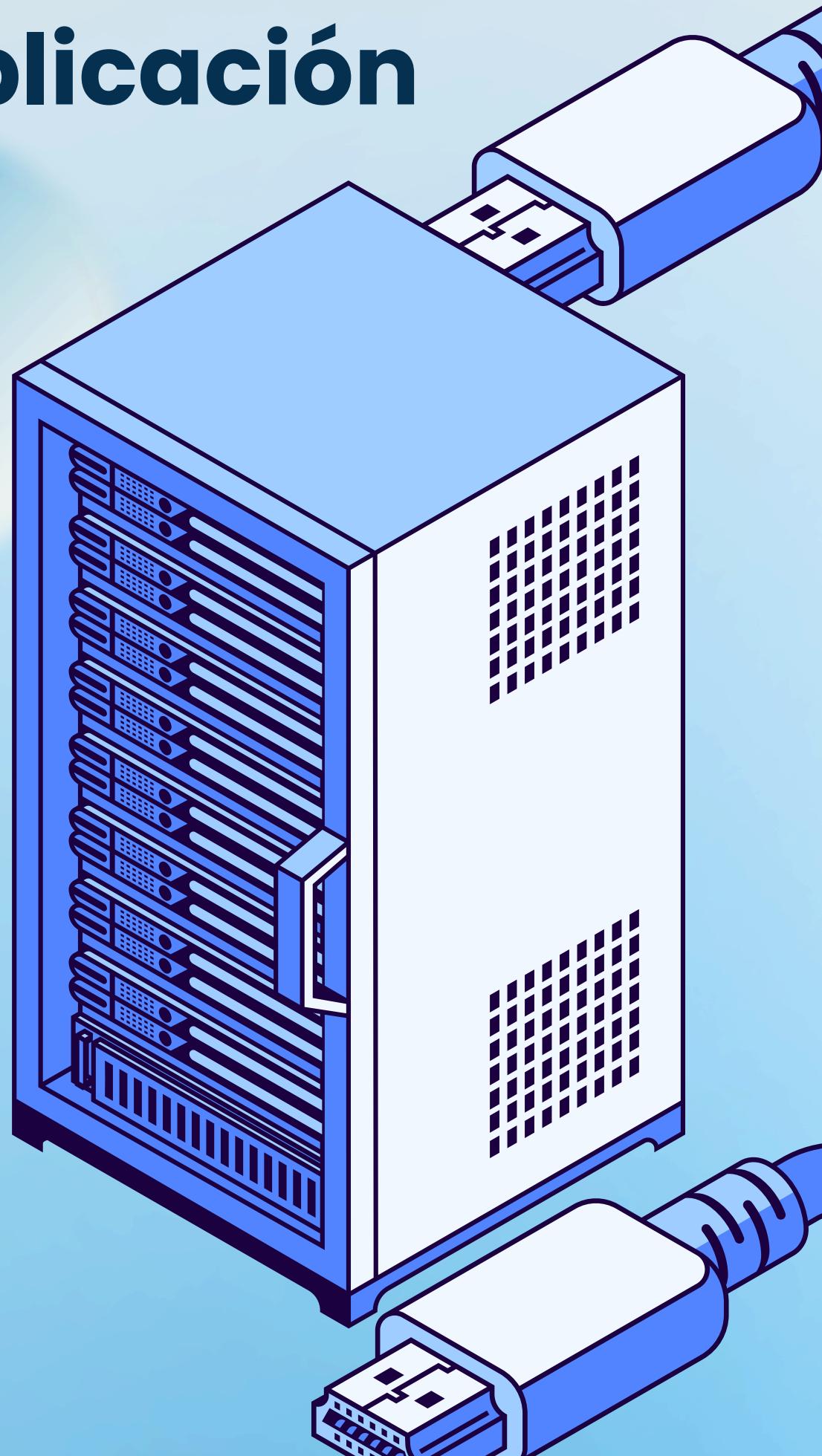
# 4. Fases del desarrollo de una aplicación

## 4.3. Codificación

En esta etapa, el programador recibe las especificaciones del diseño y las transforma en un conjunto de instrucciones escritas en un lenguaje de programación, que se conoce como **código fuente**.

Es importante respetar unas normas de codificación y estilo para que el código sea claro y homogéneo. Estas normas facilitan las tareas de corrección y mantenimiento de los programas. Por ejemplo, estos dos programas son el mismo, pero uno de ellos resulta más fácil de leer:

```
public class Ejemplo {  
    public static void main(String[] args){  
        int suma = 0;  
        int contador = 0;  
        while(contador < 10) {  
            contador++;  
            suma = suma + contador;  
        }  
        System.out.println("Suma => " + suma);  
    }  
}  
  
public class Ejemplo {  
    public static  
    void main(String[] args) {  
        int suma=0; int contador=0;  
        while(contador <10)  
        {  
            contador++;  
            suma=suma+contador;  
        }  
        System.out.println("Suma => "  
        + suma);  
    }  
}
```



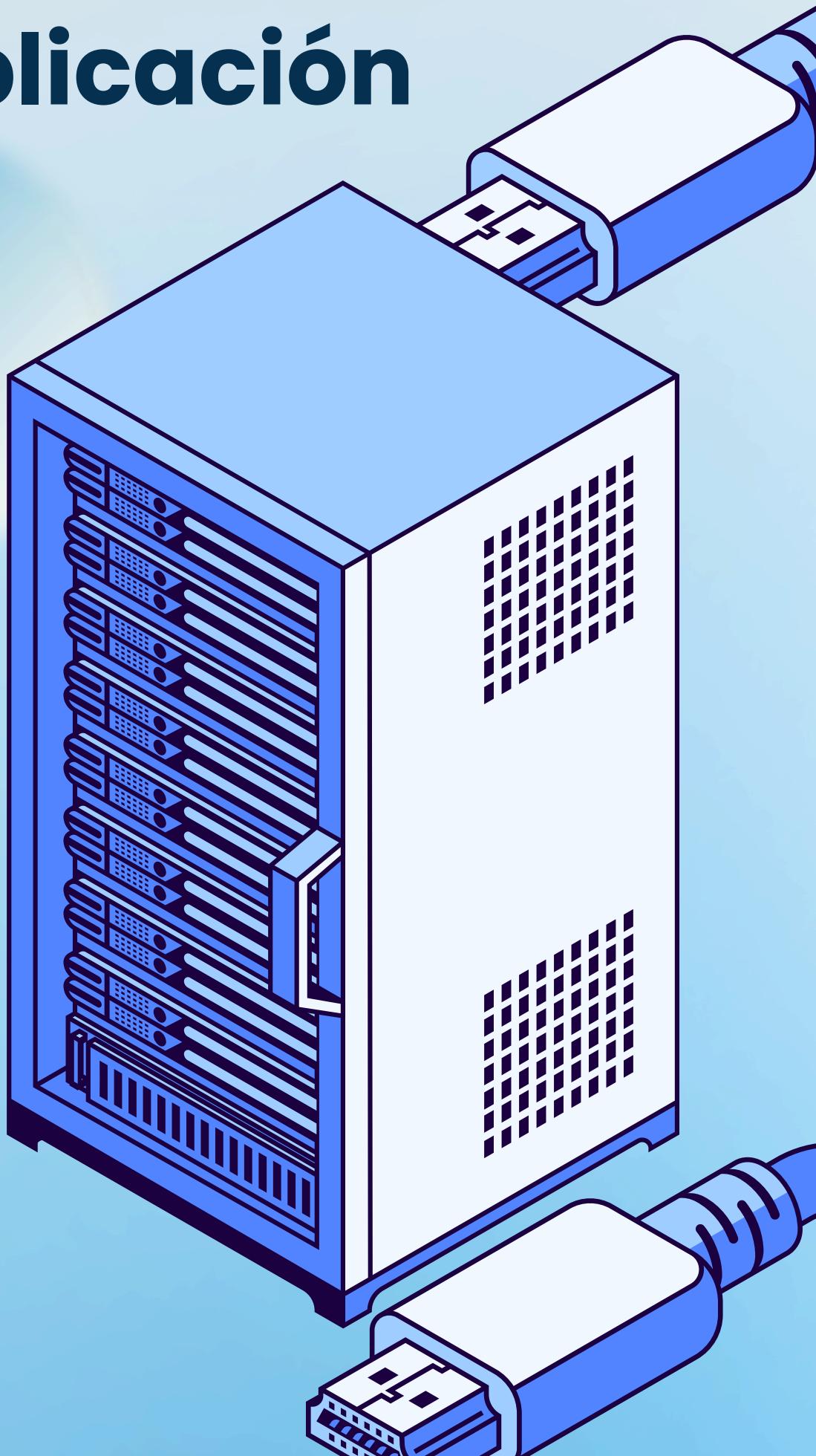
# 4. Fases del desarrollo de una aplicación

## 4.3. Codificación

A continuación se describen una serie de normas de escritura de código fuente en Java que facilita la lectura de los programas, haciendo que sea más sencillo mantenerlos y encontrar errores.

- **Nombres de ficheros:** la extensión para los ficheros de código fuente es .java y para los ficheros compilados es .class.
- **Organización de ficheros:** todos los ficheros deben comenzar con un comentario que muestre el nombre de la clase, información de la versión, la fecha y el aviso de derechos de autor.

```
/*
 * Nombre de clase
 *
 * Información de la versión
 *
 * Fecha
 *
 * Aviso de Copyright
 */
package paquete.ejemplo;
import java.io.*;
```

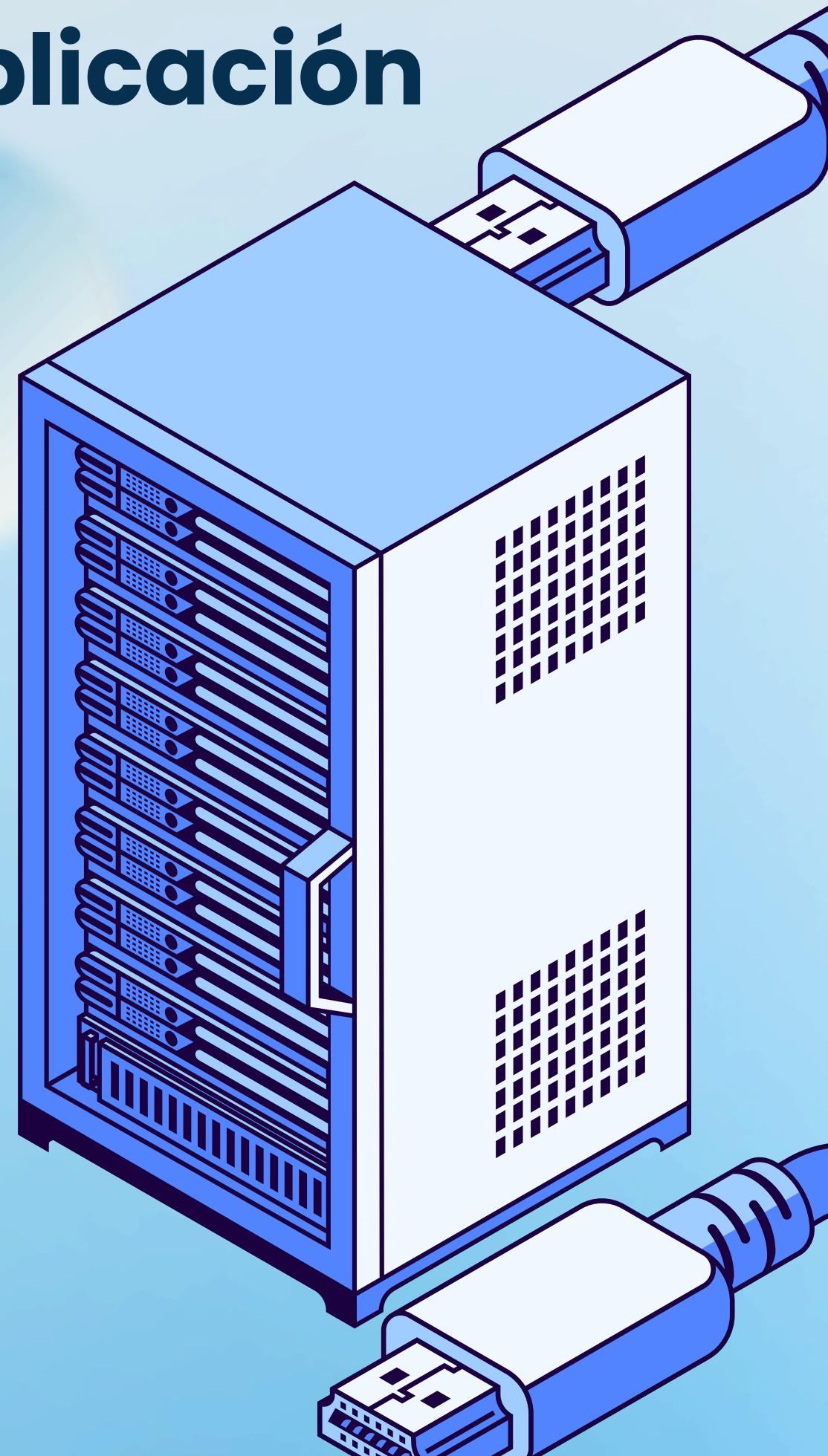


# 4. Fases del desarrollo de una aplicación

## 4.3. Codificación

- **Identación:** como norma general no se utilizarán más de 4 espacios. La longitud de las líneas de código no debe superar 80 caracteres y las líneas de comentarios no deben superar 70 caracteres. Cuando una expresión no cabe en una sola línea: romper después de una coma, romper antes de un operador y alinear la nueva línea al principio de la anterior.
- **Comentarios:** deben contener sólo información relevante para la lectura y la comprensión del programa. Existen dos tipos de comentarios: de documentación y de implementación. Los comentarios de documentación están destinados a describir la especificación del código, como por ejemplo describir las clases. Tienen el siguiente formato:

```
/**  
 * La clase Ejemplo proporciona ...  
 */  
public class Ejemplo { ... }
```



# 4. Fases del desarrollo de una aplicación

## 4.3. Codificación

Los comentarios de implementación son para comentar aspectos más específicos de la aplicación, como por ejemplo explicar estructuras de control. Puede ser de 3 tipos:

- Comentarios de bloque:  

```
/*
 * Esto es un comentario de bloque
 */
```
- Comentarios de línea: /\* Esto es un comentario de línea \*/
- Comentario corto: // Esto es un comentario corto

- **Declaraciones:** se recomienda declarar una variable por línea. Además se recomienda inicializar las variables donde están declaradas y colocarlas al comienzo del bloque. Ejemplo:

```
void miMetodo() {
    int var1 = 0;          //comienza el bloque de miMetodo
    int var2 = 10;
    if (var1 == var2) {
        int suma = 0;      // comienza el bloque if
        ....
    } else {
        var2 = var1;
        ....
    }
    ....
}
```

# 4. Fases del desarrollo de una aplicación

## 4.3. Codificación

En general, cualquier llave de apertura “{” se coloca en la misma línea que la clase o estructura de control a la que pertenece. La llave de cierre “}” aparece en una línea aparte y a la misma altura que el inicio del bloque. Los métodos o clases se separan por una línea en blanco.

- **Separaciones:** mejoran la legibilidad del código. Se utilizan dos líneas en blanco entre las definiciones de clases; y una línea en blanco entre métodos, o entre la definición de variables y la primera instrucción.
- **Nombres:** los nombres de las variables, métodos, clases, etc. hacen que los programas sean más fáciles de leer ya que pueden darnos información acerca de su función. Las normas para asignar nombres son las siguientes:
  - Clases e interfaces: los nombres deben ser sustantivos. Si el nombre está formado por varias palabras, la primera letra de cada palabra debe estar en mayúscula. Ejemplo: HiloServidor.
  - Métodos: se deben usar verbos en infinitivo. Si está formado por varias palabras, el verbo debe empezar por minúscula y la siguiente palabra en mayúscula. Ejemplo: asignarDestino().
  - Variables: deben ser cortas y la primera palabra debe estar en minúscula. Ejemplos: i, j, sumaTotal.
  - Constantes: el nombre debe ser descriptivo. Se escriben en mayúsculas y si son varias palabras van unidas por un carácter subrayado. Ejemplo: MAX\_VALOR.

Por ejemplo:

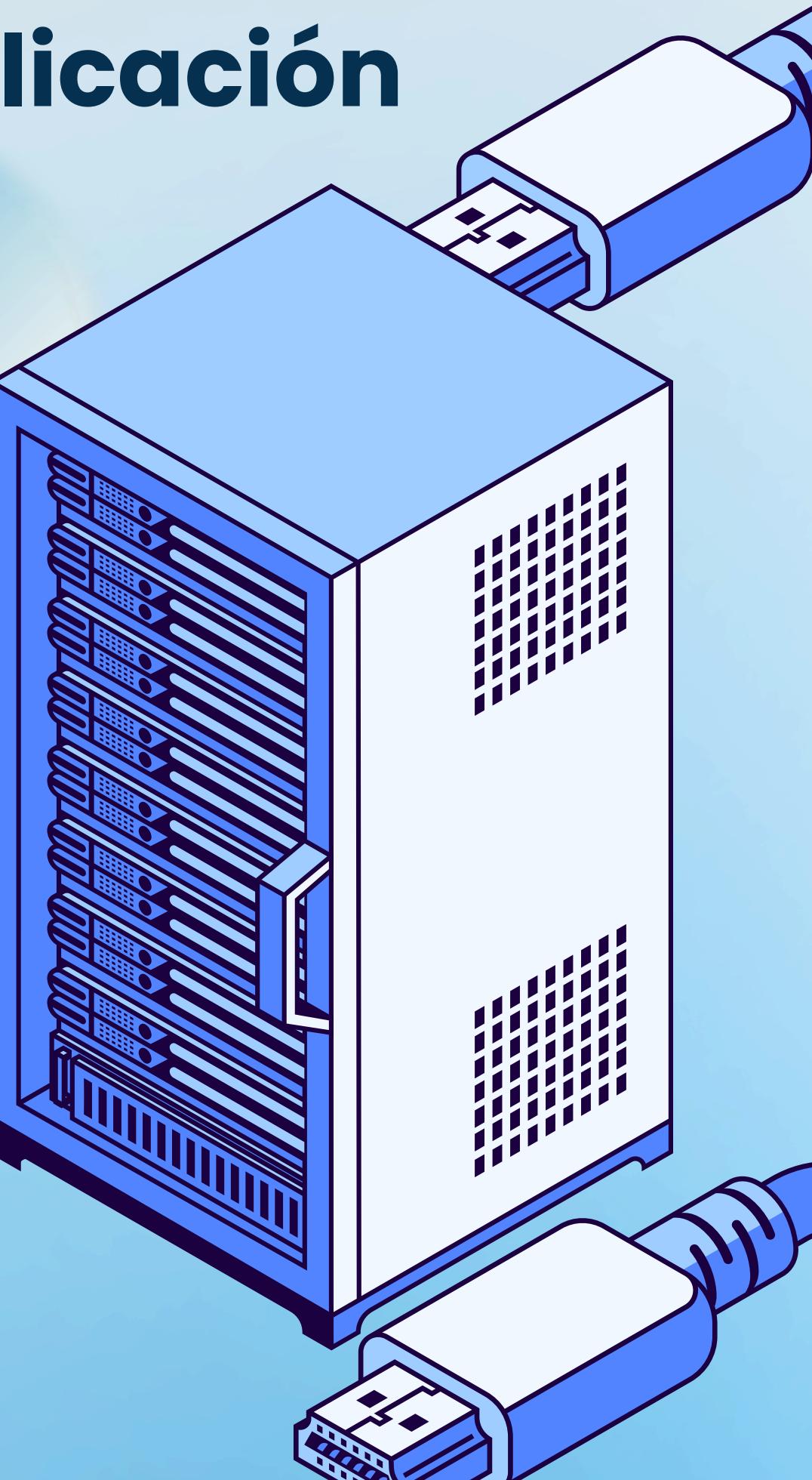
```
class Ejemplo extends Object {  
    int var1;  
    int var2;  
  
    Ejemplo(int i, int j) {  
        var1 = i;  
        var2 = j;  
    }  
  
    int metodoVacio() {}  
    ...  
}
```



# 4. Fases del desarrollo de una aplicación

## 4.3. Codificación

Las herramientas utilizadas para el desarrollo de los programas suelen ayudar a formatear correctamente el código. Prueba si es posible en VSCode.



# 4. Fases del desarrollo de una aplicación

## 4.4. Pruebas

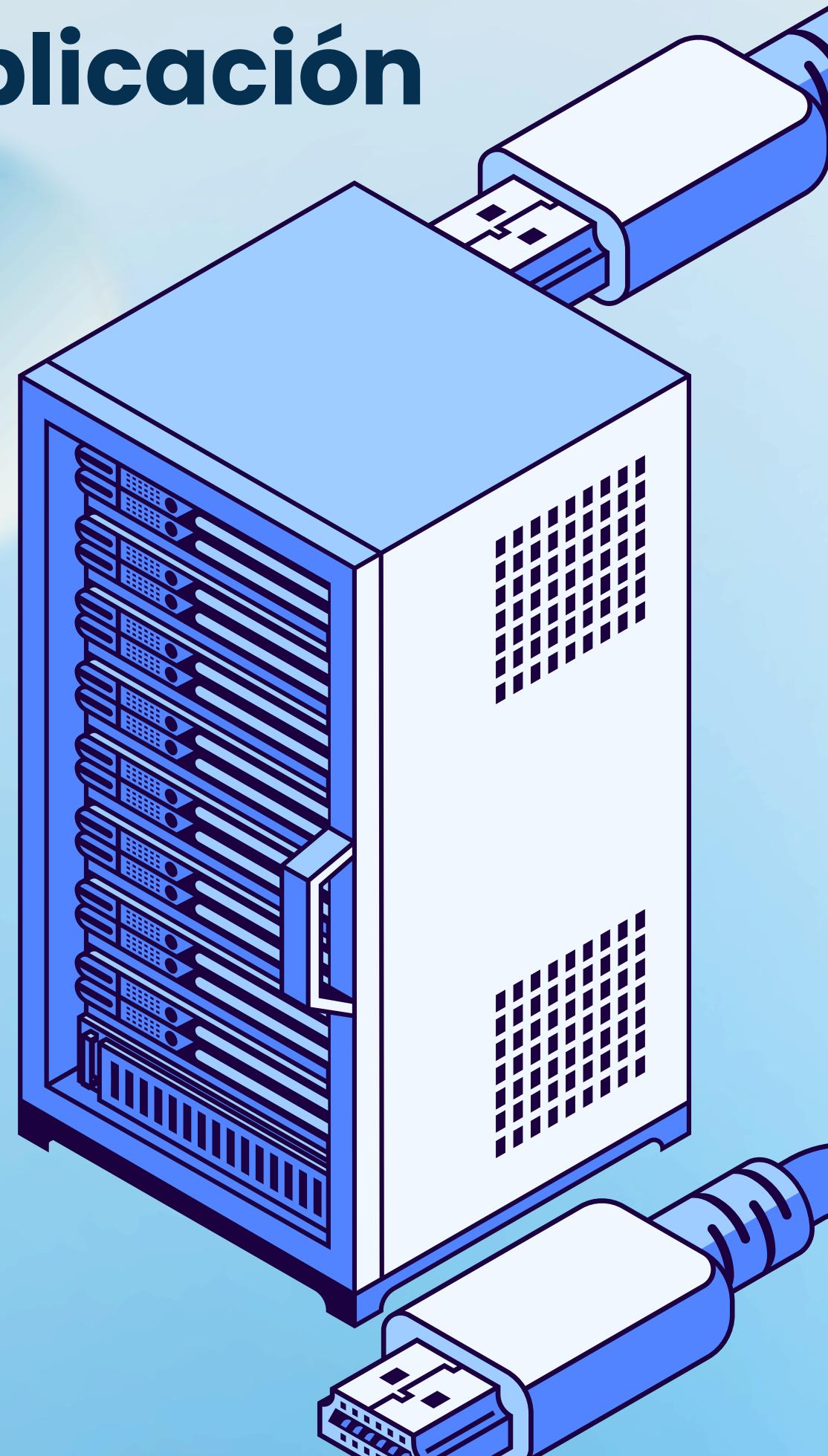
En esta etapa ya se dispone del software y se trata de encontrar errores, no solo de codificación sino también relativos a la especificación del diseño.

El objetivo de esta etapa es planificar y diseñar pruebas que sistemáticamente saquen a la luz diferentes errores. Una prueba tiene éxito si descubre un error no detectado hasta entonces.

Un **caso de prueba** es un documento que especifica los valores de entrada, salida esperada y las condiciones previas para la ejecución de la prueba.

Las recomendaciones para las pruebas son las siguientes:

- cada prueba debe definir los resultados de salida esperados.
- un programador debe evitar probar sus propios programas.
- es necesario revisar los resultados de cada prueba en profundidad.
- las pruebas deben incluir datos de entrada válidos y esperados, así como no válidos e inesperados.

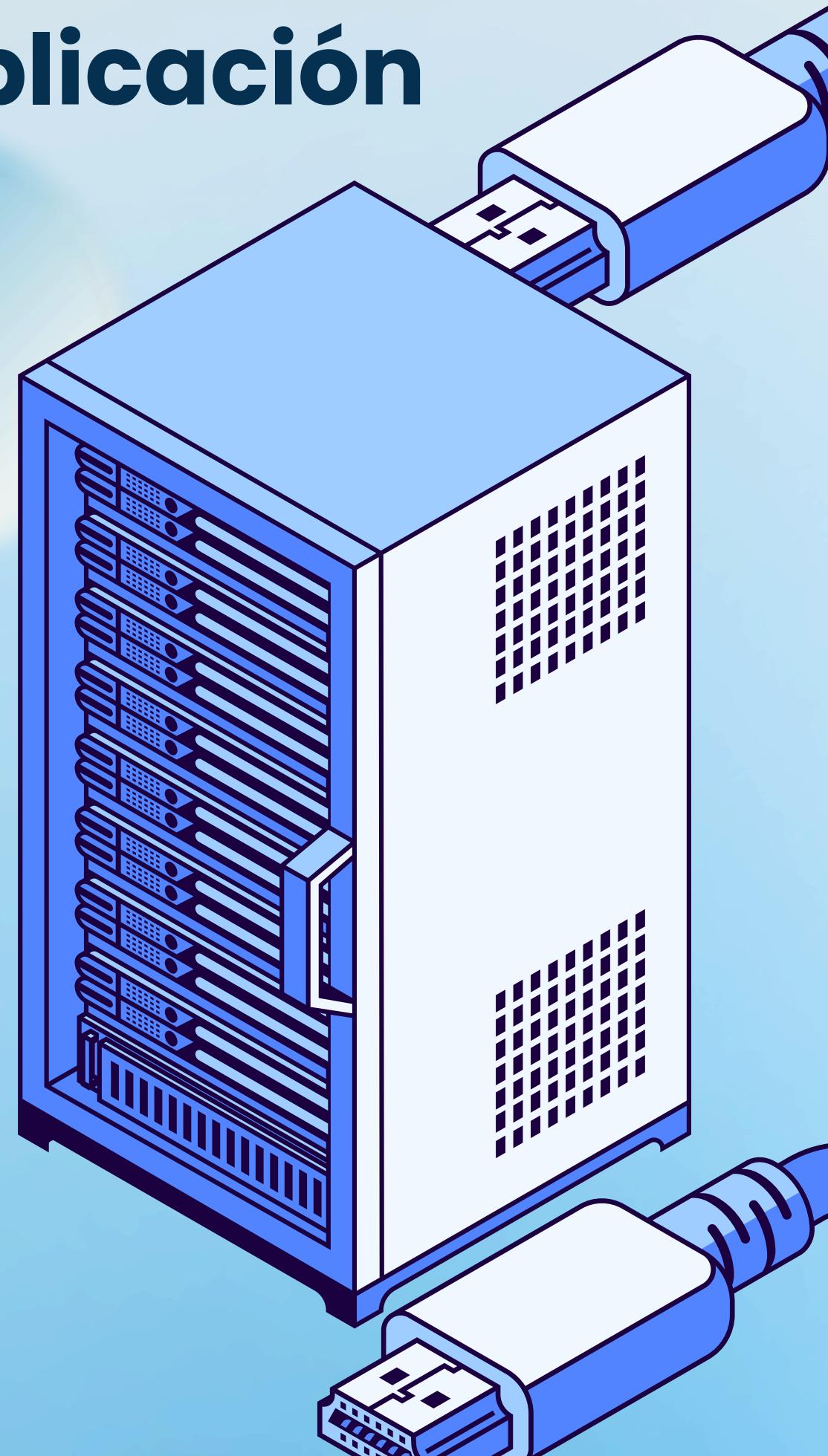


# 4. Fases del desarrollo de una aplicación

## 4.4. Pruebas

El flujo de proceso para probar el software es el siguiente:

1. Generar los casos de prueba.
2. Definir los procedimientos de la prueba, especificando cómo se va a llevar a cabo el proceso, quién lo va a realizar, cuándo, etc.
3. Ejecución de las pruebas aplicando los casos de prueba generados previamente.
4. Evaluación. Se identifican los posibles errores producidos al comparar los resultados obtenidos en la ejecución con los esperados. Es necesario realizar un informe con el resultado de ejecución de las pruebas, qué casos de prueba pasaron satisfactoriamente, cuáles no y qué fallos se detectaron.
5. Depuración. Trata de localizar y corregir los errores. Si se corrige un error se debe volver a probar el software para ver que se ha resuelto el problema.



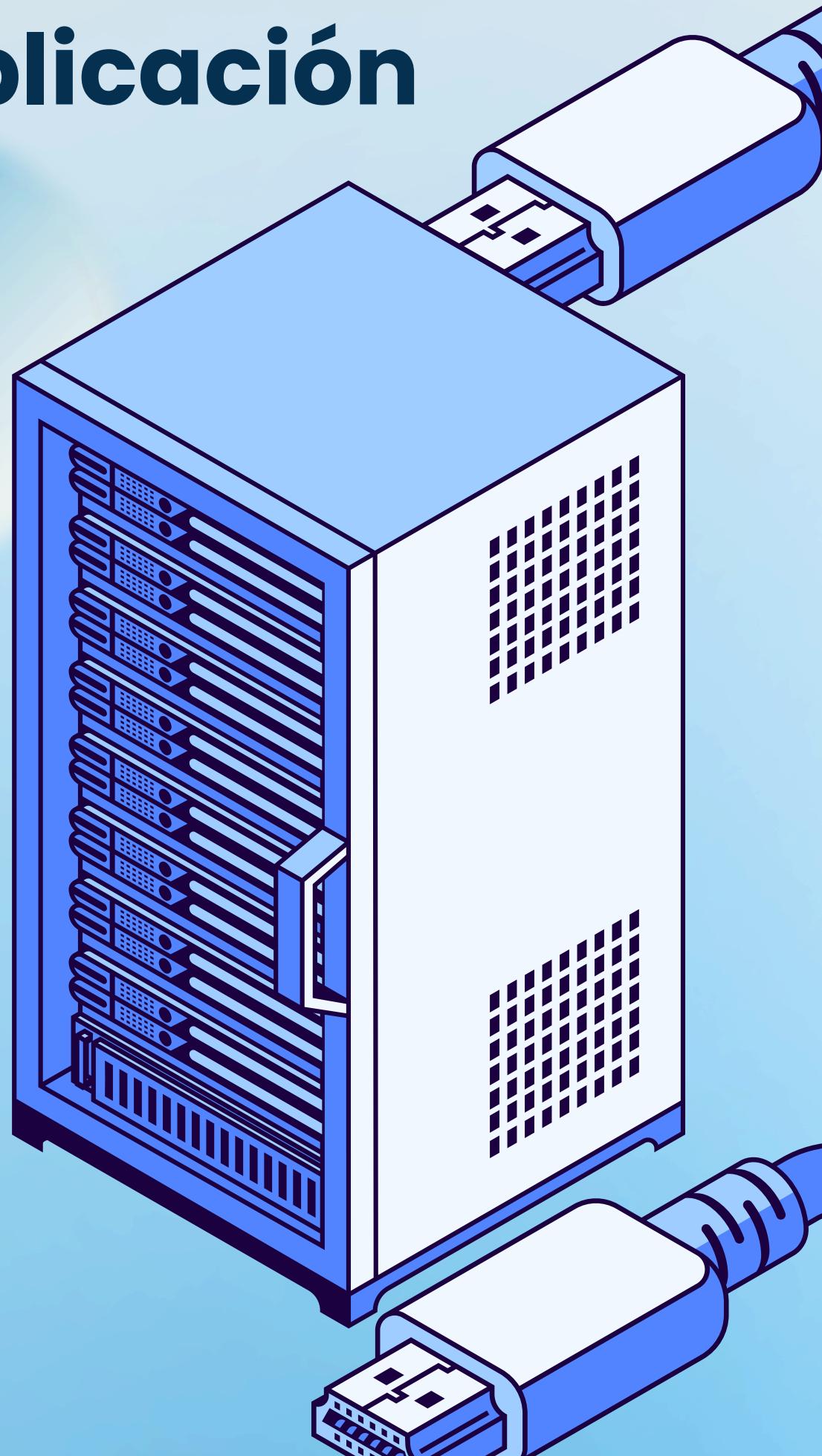
# 4. Fases del desarrollo de una aplicación

## 4.5. Documentación

Todas las etapas del desarrollo deben quedar perfectamente documentadas. En esta etapa es necesario reunir todos los documentos generados.

Los documentos relacionados con un proyecto de software:

- deben actuar como un medio de comunicación entre los miembros del equipo de desarrollo.
- deben ser un repositorio de información del sistema para ser utilizado por el personal de mantenimiento.
- deben proporcionar información para ayudar a planificar la gestión del presupuesto y programar el proceso del desarrollo del software.
- algunos de los documentos deben indicar a los usuarios cómo utilizar y administrar el sistema.



# 4. Fases del desarrollo de una aplicación

## 4.6. Explotación

Una vez que se han realizado todas las pruebas y documentado todas las etapas se pasa a la explotación del sistema. En esta etapa se lleva a cabo la instalación y puesta en marcha del producto software en el entorno de trabajo del cliente.

En esta etapa se llevan a cabo las siguientes tareas:

- **Se define la estrategia para la implementación del proceso.** Se definen los procedimientos para recibir, registrar, solucionar, hacer un seguimiento de problemas y para probar el producto software en el entorno de trabajo.
- **Pruebas de operación.** Para cada release del producto software se llevarán a cabo pruebas de funcionamiento antes de liberar el software para su uso operativo.
- **Uso operacional del sistema.** El sistema entrará en acción en el entorno previsto de acuerdo con la documentación de usuario.
- **Soporte al usuario.** Se deberá proporcionar asistencia y consultoría a los usuarios cuando la soliciten.



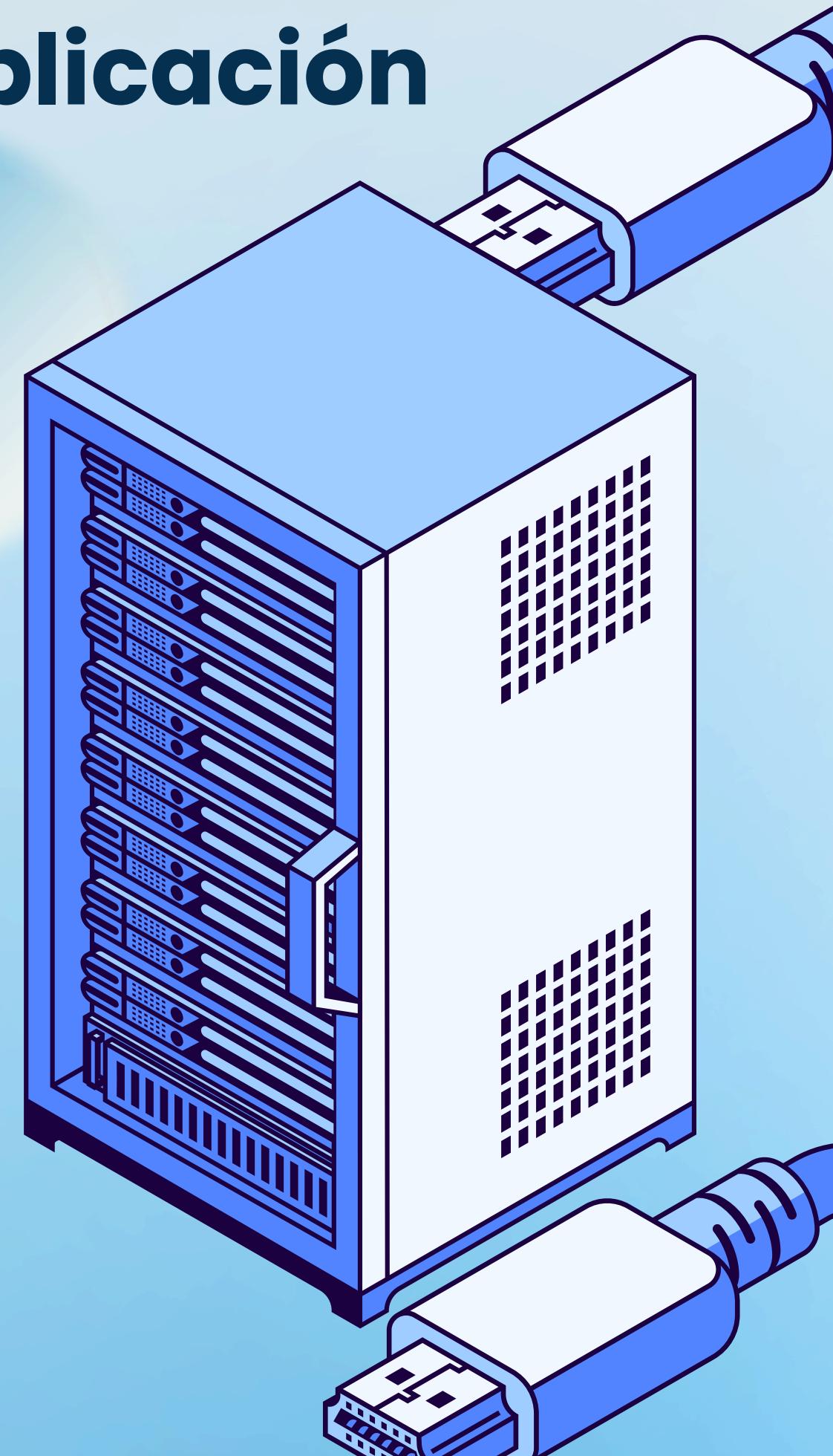
# 4. Fases del desarrollo de una aplicación

## 4.7. Mantenimiento

El mantenimiento de software se define como la modificación de un producto de software después de la entrega para corregir los fallos, para mejorar el rendimiento, o para adaptar el producto a un entorno modificado.

Existen cuatro tipos de mantenimiento del software que dependen de las demandas de los usuarios del producto software a mantener:

- **Mantenimiento adaptativo.** Con el paso del tiempo es posible que se produzcan cambios en el entorno original (CPU, sistema operativo,...). Este tipo de mantenimiento es el más usual debido a los rápidos cambios que se producen en la tecnología informática.
- **Mantenimiento correctivo.** Es muy probable que después de la entrega del producto, el cliente encuentre errores o defectos.
- **Mantenimiento perfectivo.** conforme el cliente utiliza el software puede descubrir funciones adicionales que le pueden aportar beneficios.
- **Mantenimiento preventivo.** Consiste en la modificación del producto de software sin alterar las especificaciones del mismo. Este tipo de mantenimiento hace cambios con el fin de que ser puedan corregir, adaptar y mejorar más fácilmente.



# **Capítulo 1:**

# **Desarrollo de**

# **software**

Yoshi Rosanes  
Entornos de Desarrollo  
IES Clara del Rey

