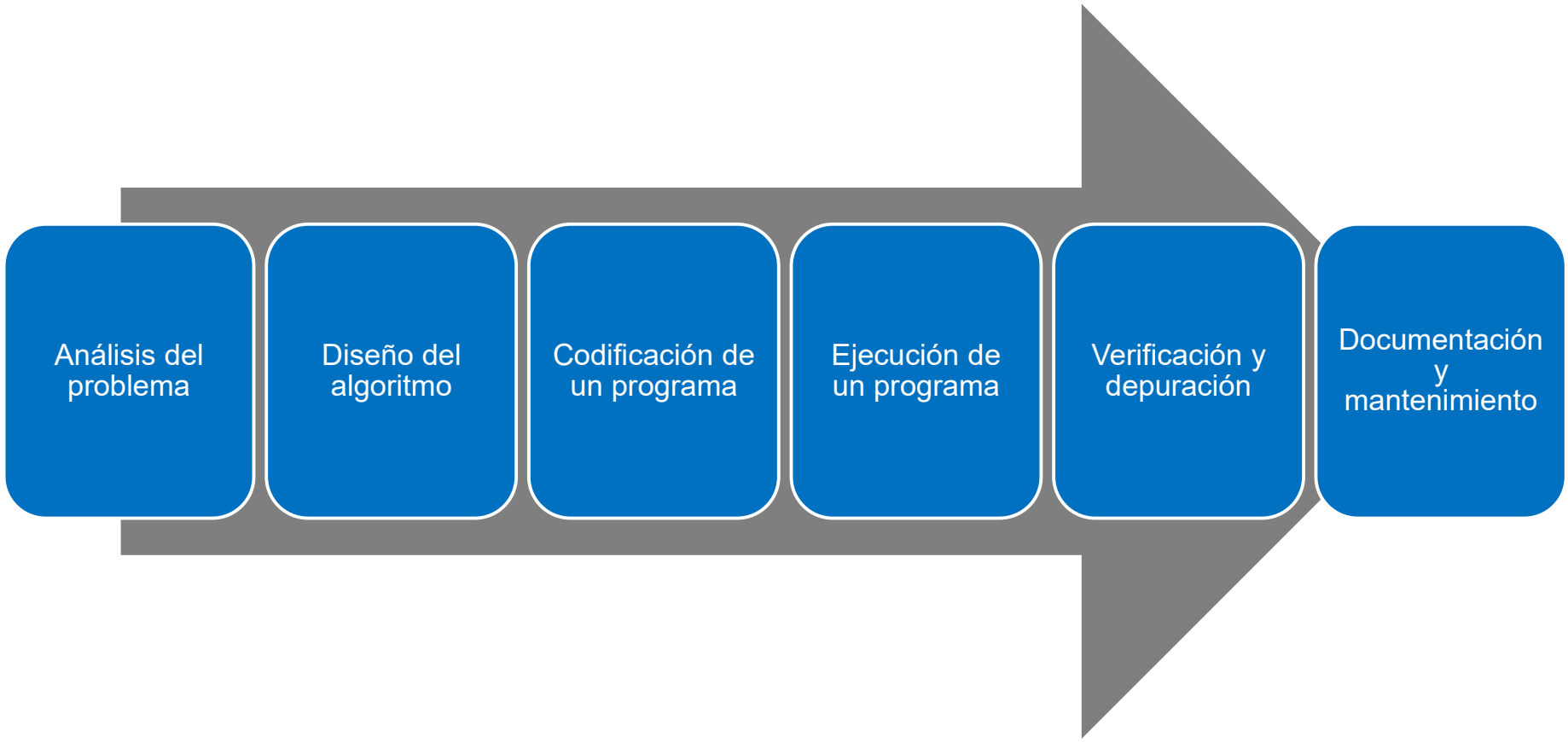
Tema 1: Fundamentos de Programación

Introducción a la programación I

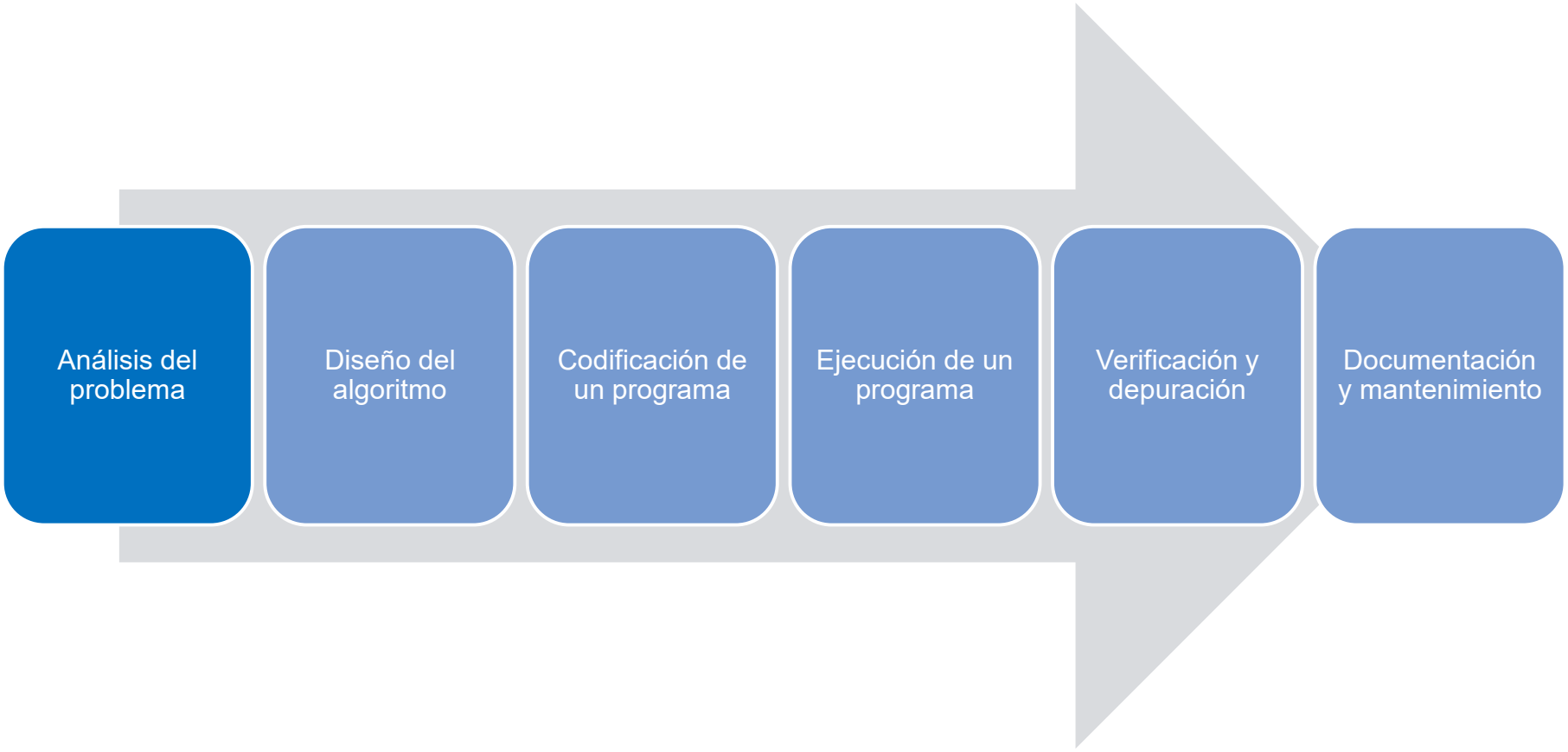
Marcos Novalbos
Elena Garcia Gamella

Índice

1. Fases en la resolución de problemas



1. 1 Análisis del Problema



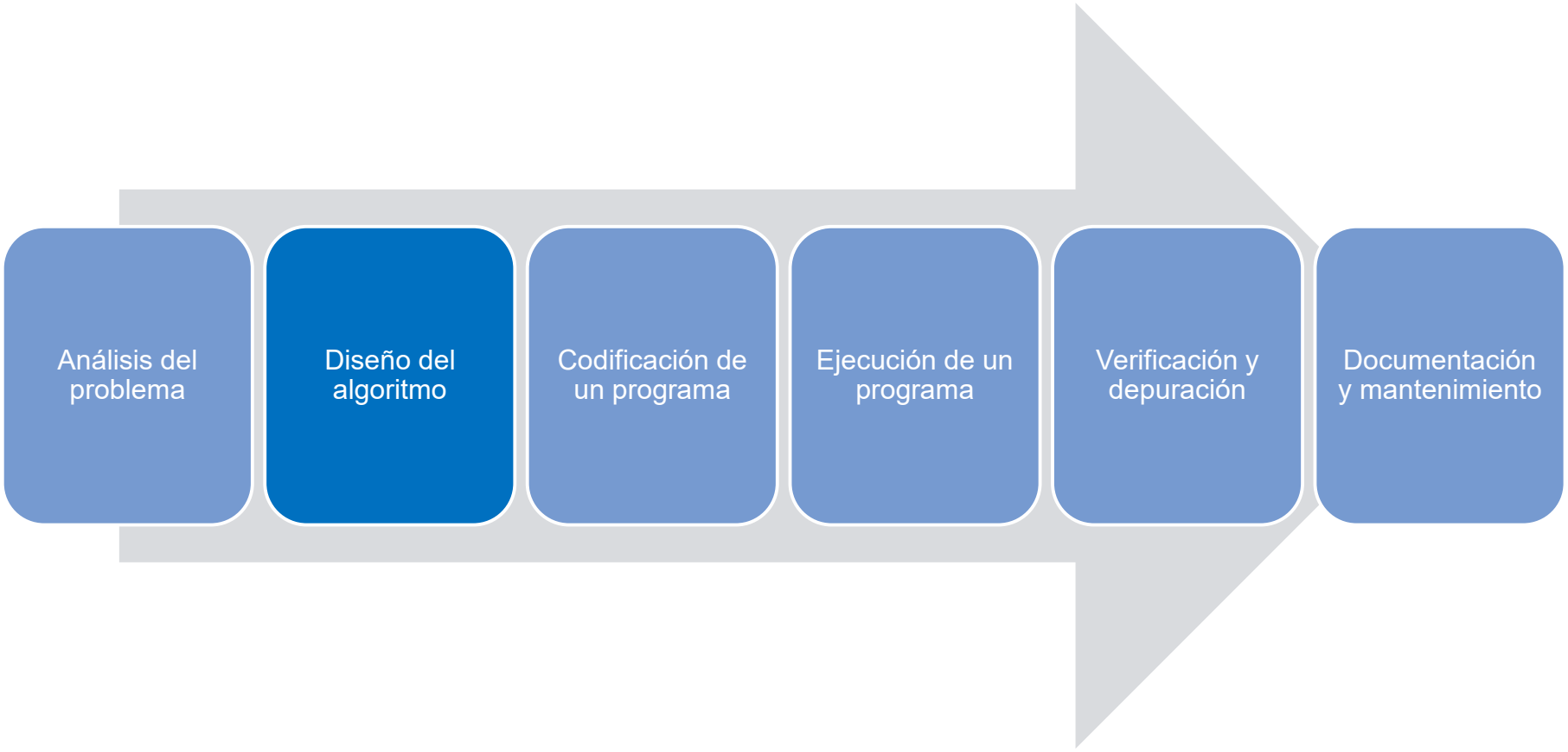
1.1 Análisis del problema

- Consiste en estudiar detalladamente el problema identificando los datos y recursos disponibles para especificar claramente lo necesario para resolverlo.

- Para hacerlo hay que responder a las siguientes preguntas:
  - ¿Qué entrada tiene el programa? (Precondiciones)
  - ¿Cuál es la salida deseada? (Postcondiciones)
  - ¿Qué método produce la salida deseada a partir de los datos de entrada? (Algoritmo)

- Esta fase comienza con el análisis de los requisitos del usuario

1. 1 Diseño del Algortimo



1.2 Diseño del algoritmo

- Un algoritmo es un conjunto finito de reglas que generan una serie de operaciones que sirven para resolver un determinado problema y que cumple las siguientes características:
  - **Finito**. Debe acabar siempre tras un número finito de pasos, si bien este número de pasos puede ser arbitrariamente grande.
  - **Está bien definido**. Cada paso del algoritmo debe definirse de modo preciso. Las acciones del algoritmo deben estar expresada sin ambigüedad.
  - **Efectivo**. Las operaciones del algoritmo deben ser básicas, estar expresadas de modo exacto, son finitas y deben ejecutar aquello que para lo que han sido definidas.
  - **Entrada**. Todo algoritmo suele tener entrada de datos pero puede no tener ningún dato de entrada.
  - **Salida**. Todo algoritmo suele tener datos de salida pero puede no generar salida.

La importancia de la descripción y los detalles para un buen diseño !!!!

1.2 Diseño del algoritmo

Algunos ejemplos: Calcular el máximo común divisor de dos números, decidir si un número es primo, calcular el mayor de una secuencia de números…

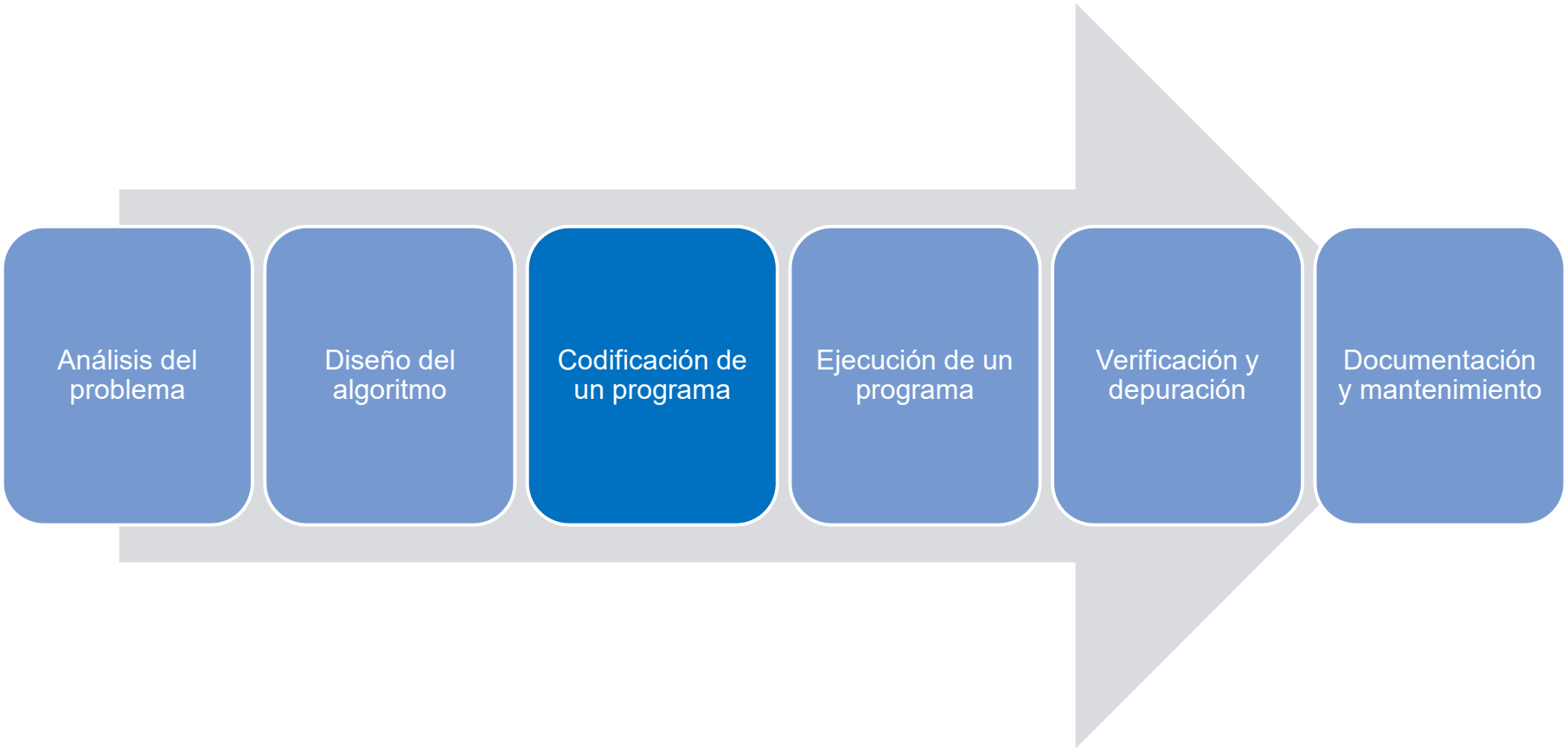
- Los métodos más eficaces se basan en el divide y vencerás.
- El gasto en tiempo en la esta fase será un ahorro cuando se escriba y depure un programa.

El resultado final del diseño es una solución que debe ser fácil de traducir a estructuras de datos y estructuras de control de un lenguaje de programación específico

1.2 Diseño del algoritmo

- Herramientas utilizadas para el diseño del algoritmo
  - Diagrama de flujo (FlowChart). Es una representación gráfica de un algoritmo. Los símbolos utilizados están normalizados (ANSI)
  - Pseudocódigo.
    - Lenguaje de especificación de algoritmos.
    - Es una herramienta de programación en la que las instrucciones se escriben en palabras que facilitan tanto la escritura como la lectura de un programa.

1. 1 Diseño del Algortimo



1.3 Codificación del programa

- Consiste en escribir las líneas de código en un determinado lenguaje de programación.
- Se siguen las siguientes reglas:
  - Si un problema se ha dividido en subproblemas, los algoritmos que resuelven cada subproblema deben ser codificados y probados independientemente.
  - Deben usarse como identificadores términos significativos usando nombres para datos y verbos para los subprogramas.
  - Ha de tenerse especial cuidado en la comunicación de los distintos subprogramas, siendo recomendable que esta comunicación se realice siempre mediante los “parámetros”.
  - El sangrado del texto (indentación), así como los buenos comentarios facilitan la posterior lectura del código.

1.3 Codificación del programa

- Un entorno de desarrollo integrado o entorno de desarrollo interactivo, en inglés Integrated Development Environment (IDE), es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.

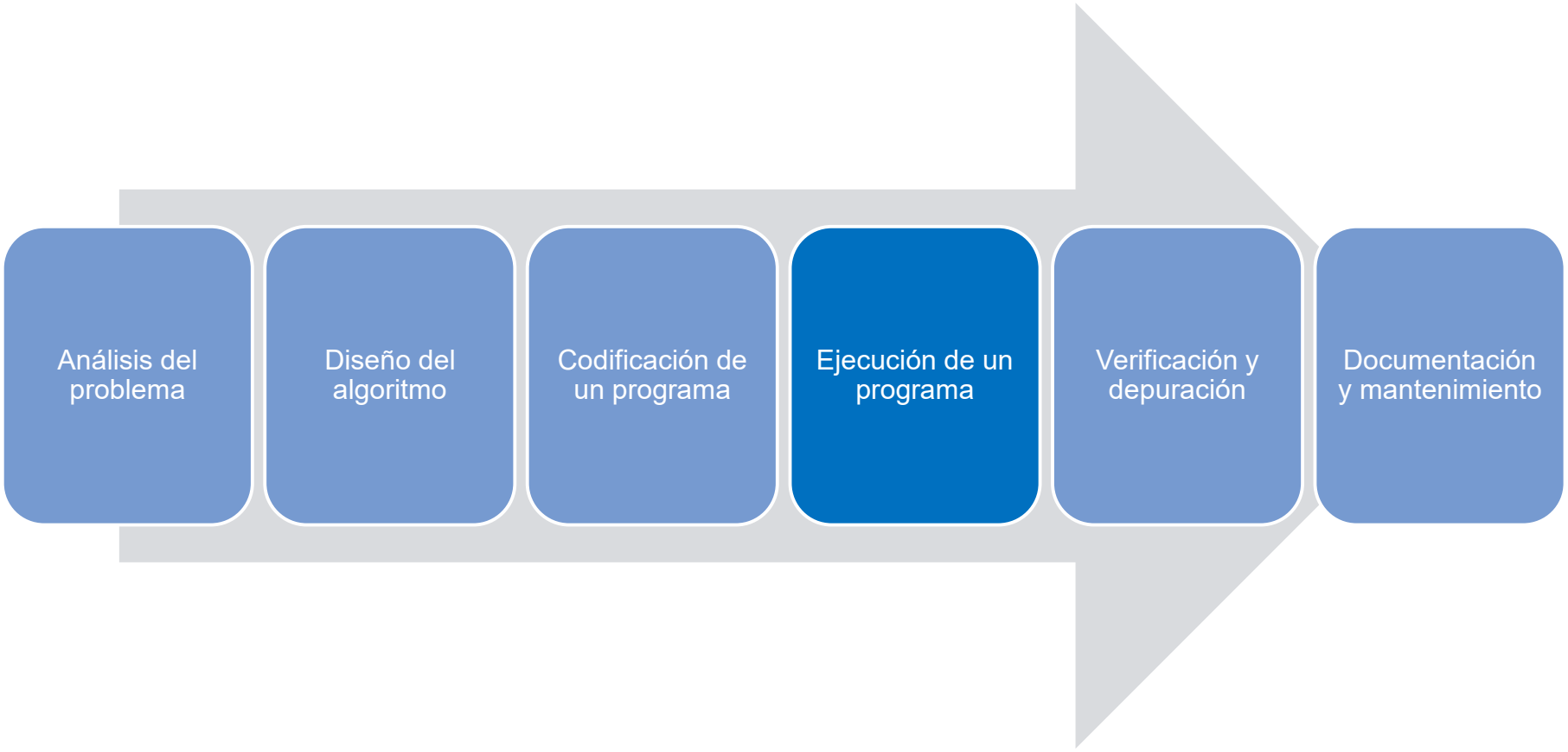
- Normalmente, un IDE consiste de un editor de código fuente, herramientas de construcción automáticas y un depurador. La mayoría de los IDE tienen auto-completado inteligente de código.

Fuente: Wikipedia

1.3 Codificación del programa

- Para programar las tareas que se desea que se ejecuten, se construyen programas de aplicación que se utilizando lenguajes de alto nivel.
- Estos programas no son interpretables por el computador, sino que están a medio camino entre el lenguaje humano y el lenguaje de los computadores.
- El ordenador sólo entiende el código binario, por lo tanto, un programa en lenguaje de alto nivel debe ser traducido o compilado al lenguaje de unos y ceros.
- Las instrucciones que interpreta y ejecuta el microprocesador se denominan instrucciones de código máquina.
- El conjunto de instrucciones distintas que puede ejecutar la CPU se denomina repertorio de instrucciones.
- El diseño del repertorio de instrucciones define las funciones que puede realizar la CPU y responden a las necesidades del programador

1. 1 Diseño del Algortimo

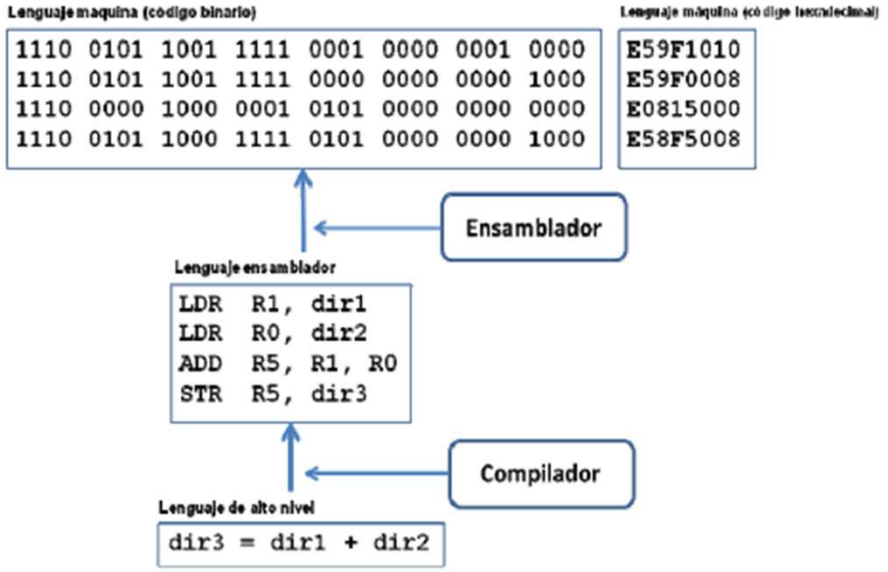


1.4 Ejecución de un programa

Modelo compilado de ejecución de programas

- El lenguaje compilado requiere un paso adicional antes de ser ejecutado, la compilación, que convierte el código que escribes a lenguaje ejecutable.

- Ejemplos: C, C++

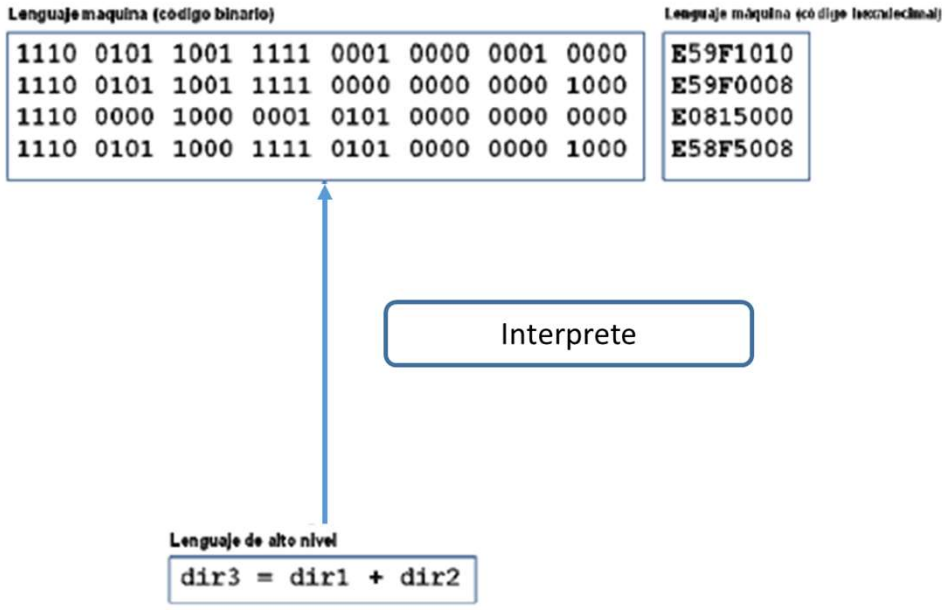


1.4 Ejecución de un programa

Modelo interpretado de ejecución de programas

- Un lenguaje interpretado es convertido a lenguaje de máquina a medida que es ejecutado. Esta función es realizada por un programa denominado interprete.

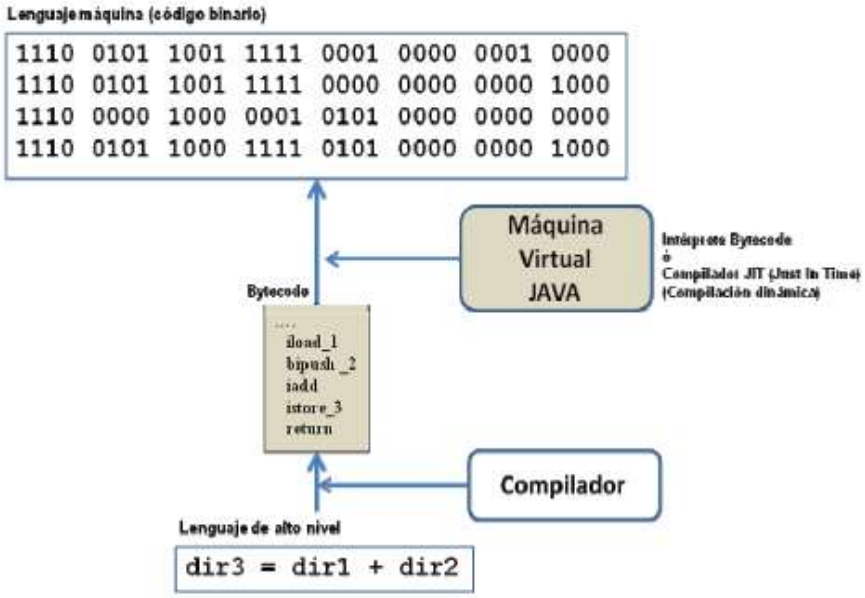
- Ejemplos: PHP, java script, Python, ..



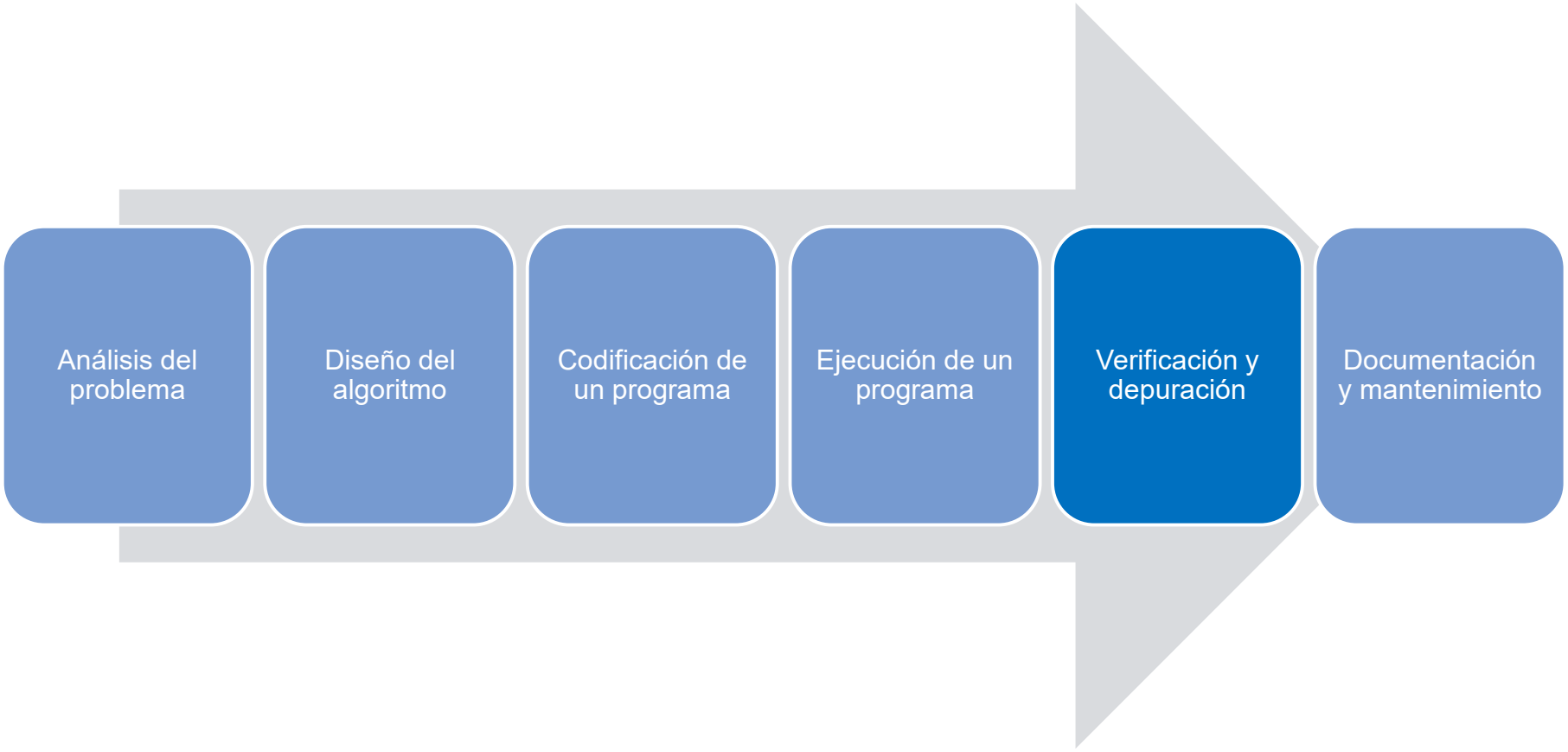
1.4 Ejecución de un programa

Modelo “hibrido” de ejecución de programas

- Cuando se compila un programa en Java, en realidad éste no se compila a lenguaje máquina, directamente entendible por el sistema operativo, sino a un lenguaje intermedio denominado Byte Code.
- Entre el Byte Code y el sistema operativo debe haber un componente especial llamado Máquina virtual de Java que es el que realmente va a ejecutar el código
- En el caso de Java, la Java Virtual Machine o JVM toma el código ByteCode resultante de compilar la aplicación Java y lo "Interpreta" a su vez a código nativo de la plataforma en la que se está ejecutando. La ventaja principal de este esquema es que es muy fácil crear un programa en Java y que luego éste se pueda ejecutar en cualquier sistema operativo para el cual exista una implementación de la JVM (hoy en día, casi literalmente todos)
- Ejemplo: Java



1. 1 Diseño del Algortimo



1.5 Depuración y Verificación

Depuración

- La depuración de un programa es el proceso de encontrar los errores de programa y corregir o eliminar dichos errores.

- **Depuración manual**
  - Se proporciona al programa entradas válidas que conducen a una solución conocida.
  - También deben incluirse datos no válidos para comprobar la capacidad de detección de errores del programa
  - Se incluyen "trazas" en el programa para ir comprobando los valores intermedios que se van obteniendo son los esperados.

1.5 Depuración y Verificación

Depuración

- **Depuración a partir de herramientas**:
  - Dada la complejidad y el tamaño de los programas para ahorrar tiempo y recursos se crearon las herramientas de depuración (debugger).
  - Todos los IDE tienen asociados su herramienta de depuración.
  - Las acciones más habituales que realizan estas herramientas son:
    - Ejecutar paso a paso un programa (stepping).
    - Marcar puntos de parada (breakpoints).
    - Examinar el contenido de las variables y objetos.
    - Conocer el encadenamiento de llamadas de procedimientos.
    - Retomar la ejecución hasta un nuevo punto de detención.

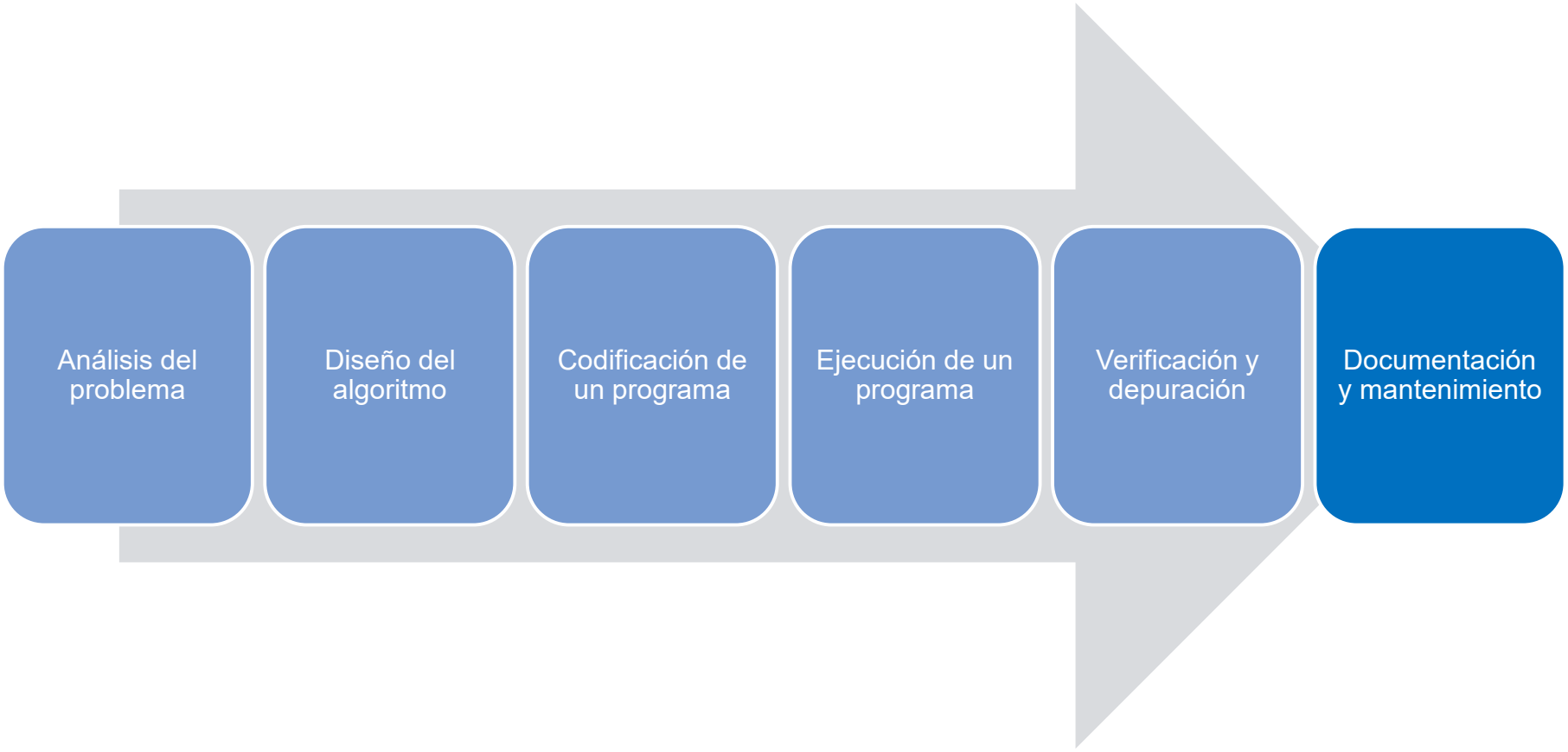
1.5 Depuración y Verificación

Verificación

- La verificación es la acción de comprobar que el programa está de acuerdo con su especificación.

- Se comprueba que el sistema cumple los requerimientos funcionales y no funcionales que se han especificado.

1. 1 Diseño del Algortimo



1.6 Documentación y Mantenimiento

Documentación

- La documentación de un programa consiste en detallar en forma de documentación los pasos que hay que realizar para resolver el problema.

- Puede ser:
  - Interna: Contenida en entre las líneas de código como comentarios.
  - Externa: Contiene información (documentos) del análisis, diagramas de flujo, pseudocódigo, manuales de usuario con instrucciones para ejecutar el programa

1.6 Documentación y Mantenimiento

Mantenimiento

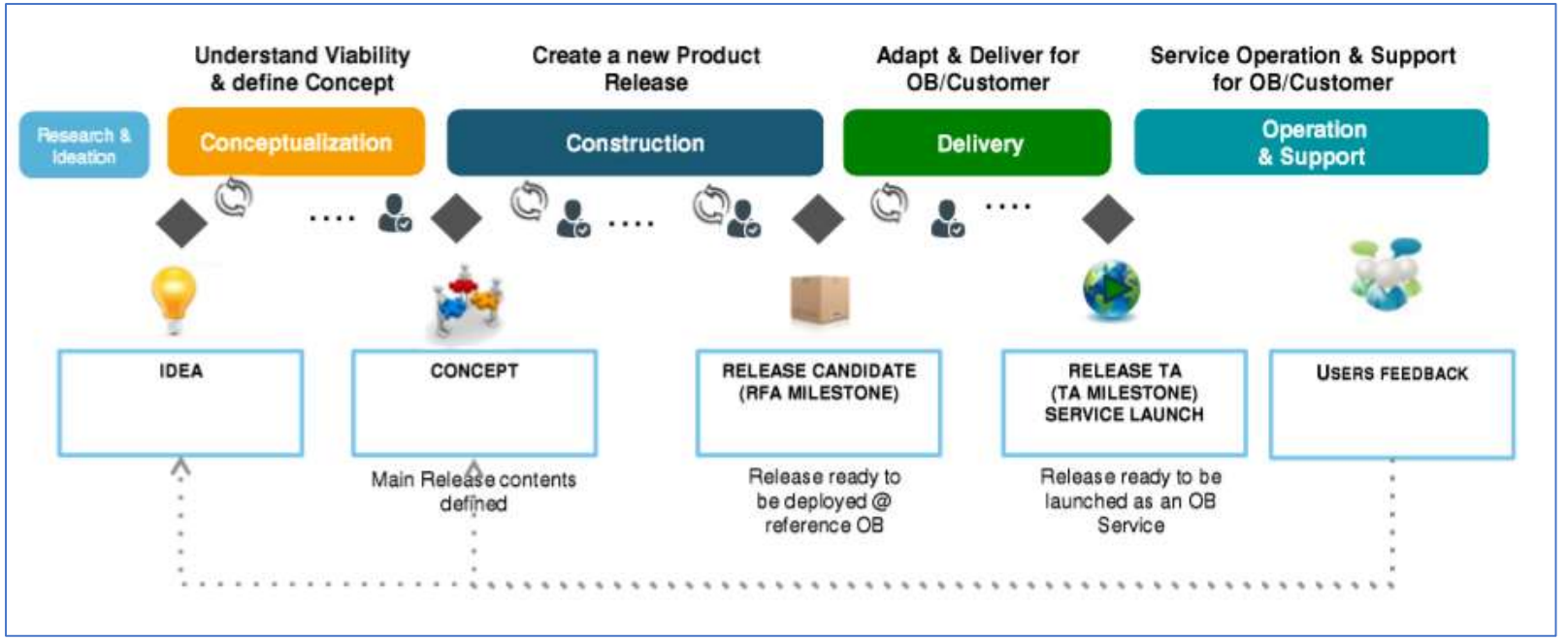
- Se entiende por **mantenimiento** de un software dos tipos de tareas fundamentalmente:
  - Las tareas de reparación/modificación: de los posibles errores detectados cuando el software ya está funcionando.
  - Las tareas de mejora y evolución: que permiten modificar atributos, o capacidades de rendimiento e incluso añadir nuevas funcionalidades.

1.7. Ciclo de desarrollo Software

Fases Generales del Ciclo de Desarrollo software:

1.7. Ciclo de desarrollo Software

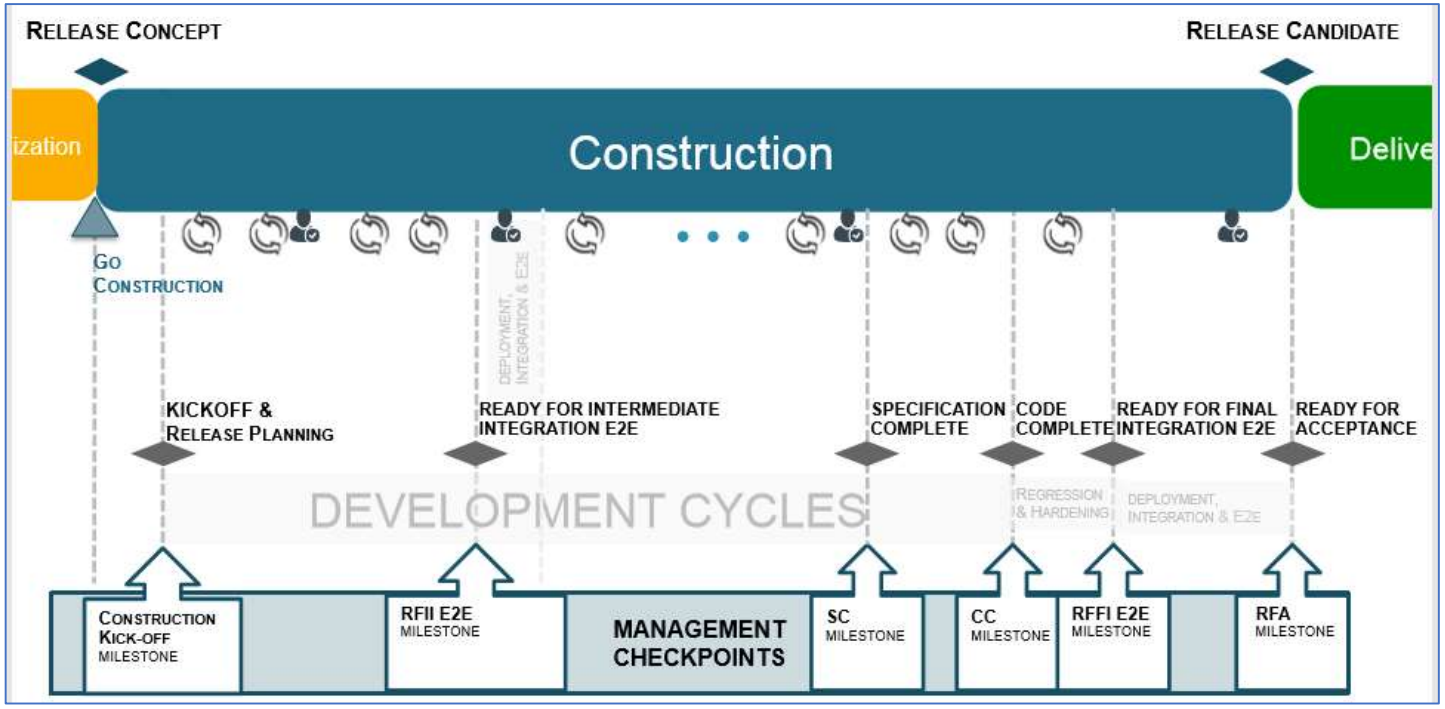
Ejemplo real del ciclo de desarrollo software en el mundo empresarial:



La mayoría de las fases del ciclo de desarrollo software son a su vez fases iterativas (aplicación de metodoligías ágiles)

1.7. Ciclo de desarrollo Software

La fase de "construcción" incluye los ciclos de desarrollo y pruebas del programa o producto



2. Programación estructurada

- La programación estructurada es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa utilizando subrutinas (funciones) y tres tipos de estructuras de control: secuencial, condicional y repetitiva.

- La programación estructurada consiste por tanto en escribir un programa de acuerdo con las siguientes reglas:

  - El programa debe ser dividido (según su complejidad) en diferentes módulos

  - Cada módulo de programa tiene un diseño modular

  - Los módulos son diseñados descendentemente

  - Cada módulo de programa se codifica usando tres tipos de estructuras de control:
    - Secuencia
    - Selección (alternativa)
    - Iteración (repetitiva)

2.1 Recursos abstractos

- Descomponer un programa en términos de recursos abstractos consiste en descomponer acciones complejas en términos de acciones más simples

- La abstracción procedimental separa el propósito de un programa, o un módulo del mismo, de su implementación.

- Una vez que el programa o el módulo del mismo, ha sido codificado se puede usar sin necesidad de conocer su cuerpo, y basta con su nombre y una descripción de sus parámetros.

2.2 Diseño descendiente (top down)

- El problema se descompone en etapas o estructuras jerárquicas.
- El resultado de esta jerarquía de módulos es que cada módulo se refina por los de nivel inferior que resuelven problemas más pequeños y contienen más detalles sobre los mismos

2.3 Estructuras de control

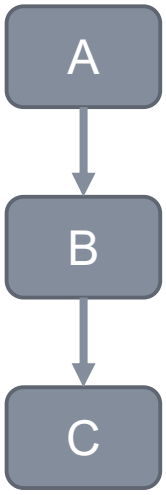
- **Las estructuras de control sirven para especificar el orden en que se ejecutarán las distintas instrucciones de un algoritmo**.

- Este orden de ejecución determina el flujo de control del programa.

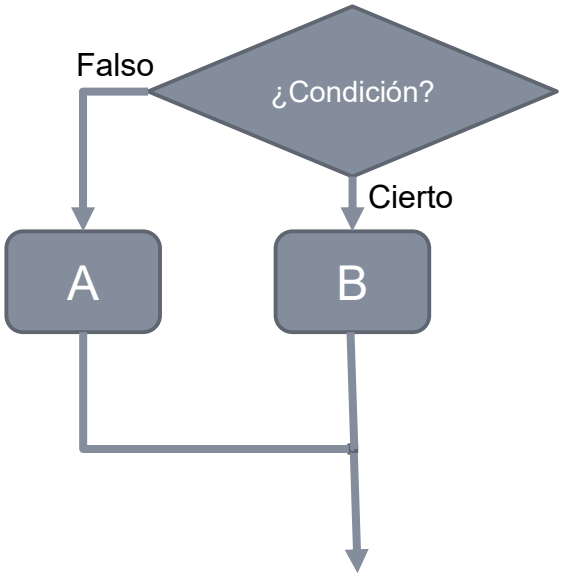
- La programación estructurada hace los programas más fáciles de escribir, verificar, leer y mantener, utilizando un número limitado de estructuras de control que minimizan la complejidad de los problemas.

2.3 Estructuras de control

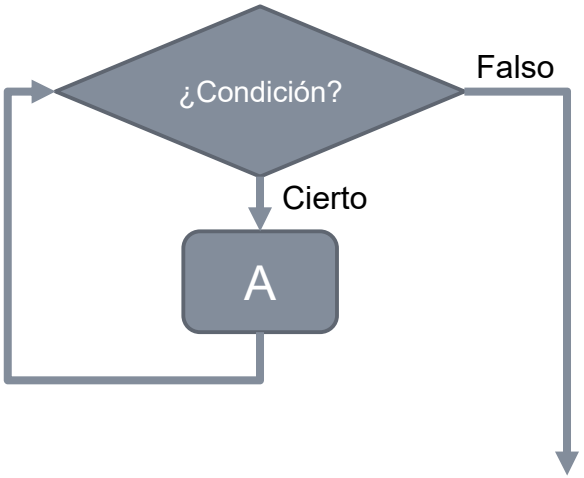
Secuencial



Alternativa o selección



Iteración o Repetitiva



Ejemplo1: algoritmos (pseudocódigo)

- Escribir el pseudocódigo de un algoritmo que lea tres números enteros y si el primero es positivo calcule el producto de los otros dos y en otro caso calcule la suma

Ejemplo1: algoritmos

- Escribir el pseudocódigo de un algoritmo que lea tres números enteros y si el primero es positivo calcule el producto de los otros dos y en otro caso calcule la suma

  - Análisis del problema
    - Se usan tres variables, Numero1, Numero2 y Numero3, que almacenan los datos de entrada leídos y otras dos variables Producto y Suma en las que se deposita el resultado de la suma.
  - Algoritmo

```
Algoritmo Producto_o_Suma
Variables
Entero Numero1, Numero 2, Numero3, Producto, Suma
Inicio
Leer (Numero1, Numero2, Numero3)
   Si (Numero > 0) entonces
       Producto <- Numero2 * Numer3
       Escribe ("El producto de los dos últimos números es", Producto)
   Si no
       Suma <- Numero2 + Numero3
       Escribe ("La suma de los dos últimos números es", Suma)
   Fin si
Fin
```

Ejemplo2: algoritmos (pseudocódigo)

- Escribir el pseudocódigo de un algoritmo que sume los 50 primeros números naturales

Ejemplo2: algoritmos (pseudocódigo)

- Escribir el pseudocódigo de un algoritmo que sume los 50 primeros números naturales

  - Análisis del problema
    - Se usa una variable `Contador` que cuenta los 50 primeros naturales y una variable `Suma`, para almacenar las sucesivas sumas 1, 1+2, 1+2+3, ….
  - Algoritmo

```
Algoritmo Suma_Cincuenta
Variables
Entero Contador, Suma
Inicio
Contador <- 0
Mientras Conatador <= 50 hacer
    Suma <- Suma + Contador
    Contador <- Contador + 1
Fin mientras
Escribe (Suma)
Fin
```


U-tad
CENTRO UNIVERSITARIO
DE TECNOLOGÍA Y ARTE DIGITAL