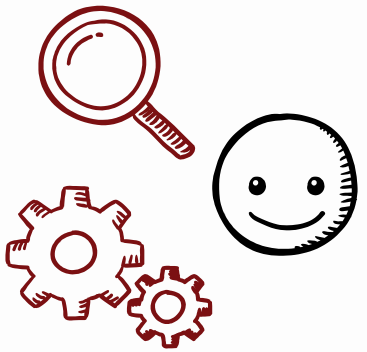
Ingeniería en
DESARROLLO
DE SOFTWARE

Programación Estructurada

Ingeniería en
DESARROLLO
DE SOFTWARE

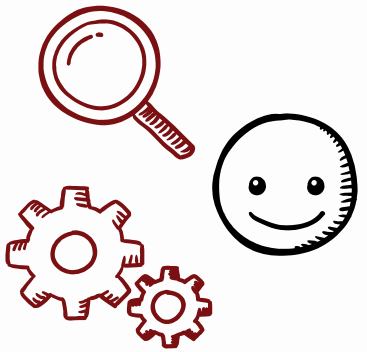
La **programación estructurada** es una **teoría orientada a mejorar la claridad, calidad y tiempo de desarrollo utilizando únicamente subrutinas o funciones**.

Basada en el teorema del programa estructurado propuesto por Böhm y Jacopini, ha permitido desarrollar software de fácil comprensión.



Ingeniería en
DESARROLLO
DE SOFTWARE

La **programación estructurada** es una corriente que nació con la vocación de facilitar la vida de los programadores, sobre todo cuando estos debían abordar fases de mejora posteriores a la creación del programa, y de ordenar la forma en la que se creaba cualquier tipo de programa. Para comprenderlo mejor, vamos a hacer un pequeño viaje en el tiempo.



Ingeniería en
DESARROLLO
DE SOFTWARE

Nos vamos al año 1966, cuando **Böhm y Jacopini** proponen el **teorema del programa estructurado**, con el que demuestran que cualquier programa puede ser escrito utilizando solo tres instrucciones de control.

Imagínate, ¡esto fue toda una revolución! Implicaba la construcción de programas más sencillos y más rápidos, en los que disminuye la complejidad de las pruebas y el testing para ponerlos en funcionamiento.

Ingeniería en DESARROLLO DE SOFTWARE

En 1968, **Edsger Dijkstra** publicó un célebre artículo que impactó en la computación moderna: **Go To Statement Considered Harmful**.

¿Por qué es tan importante? Pues porque este científico holandés promovió activamente el uso de lenguajes de programación estructurada, fomentando la verificación formal de programas y la eliminación de la sentencia Goto.

De hecho, Dijkstra participó en el comité que diseñó **Algol 60**, el primer lenguaje de programación estructurado.

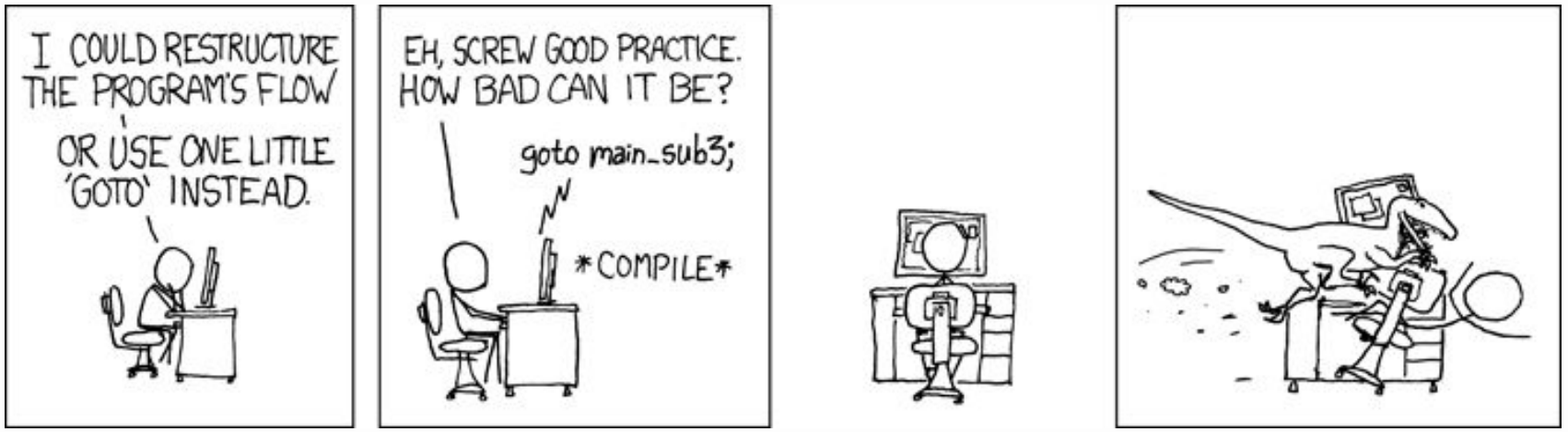
Ingeniería en DESARROLLO DE SOFTWARE

La **programación estructurada** se convierte así, junto con la programación orientada a objetos, en **uno de los paradigmas de programación más populares** que ejecuta los lenguajes más potentes que seguro conoces, incluidos, entre otros, Java, C, Python y C++.

Sin embargo, algunos lenguajes de programación más antiguos (como Fortran) se apoyaban en una sola instrucción para modificar la secuencia de ejecución de las instrucciones mediante una transferencia incondicional de su control (con la instrucción goto, del inglés "go to", que significa "ir a").

Ingeniería en DESARROLLO DE SOFTWARE

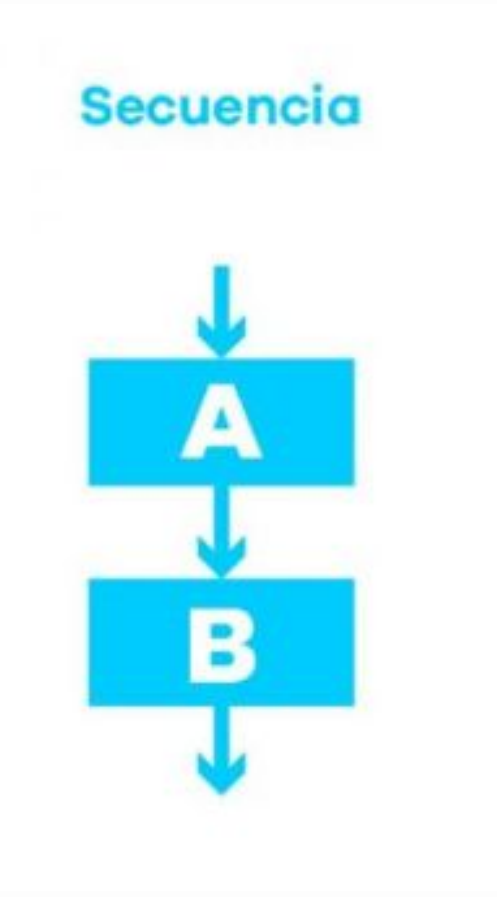
Pero estas transferencias arbitrarias del control de ejecución hacen los programas muy poco legibles y difíciles de comprender.



A finales de los años sesenta,
surgió una nueva forma de programar que reduce a la mínima expresión el uso de la instrucción goto y la sustituye por otras más comprensibles.

Ingeniería en DESARROLLO DE SOFTWARE

Esta forma de programar se basa en **un famoso teorema, desarrollado por Edsger Dijkstra**, que demuestra que **todo programa puede escribirse utilizando únicamente las tres estruc básicas de control** siguientes:



1. **Secuencia**: el bloque secuencial de instrucciones, instrucciones ejecutadas sucesivamente, una detrás de otra.

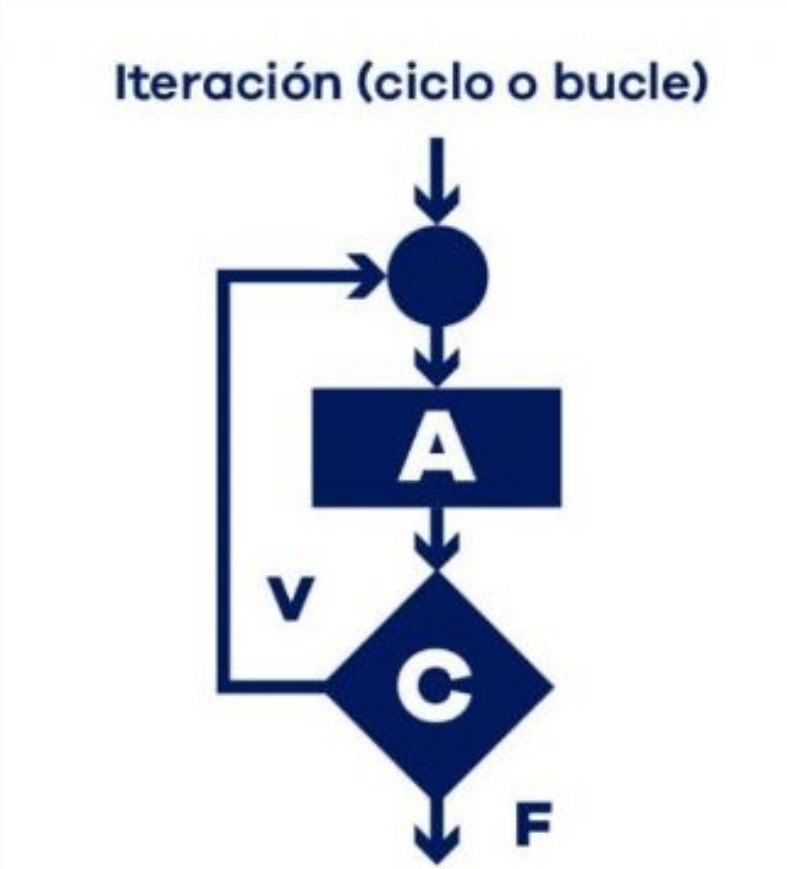
Ingeniería en
DESARROLLO
DE SOFTWARE

2. **Selección**: la instrucción condicional con doble alternativa, de la forma:

"if **condición** then **instrucción-1** else **instrucción-2".**

Ingeniería en DESARROLLO DE SOFTWARE

3. **Iteración**: el bucle condicional "while condición do instrucción", que **ejecuta** la instrucción repetidamente mientras la condición se cumpla.

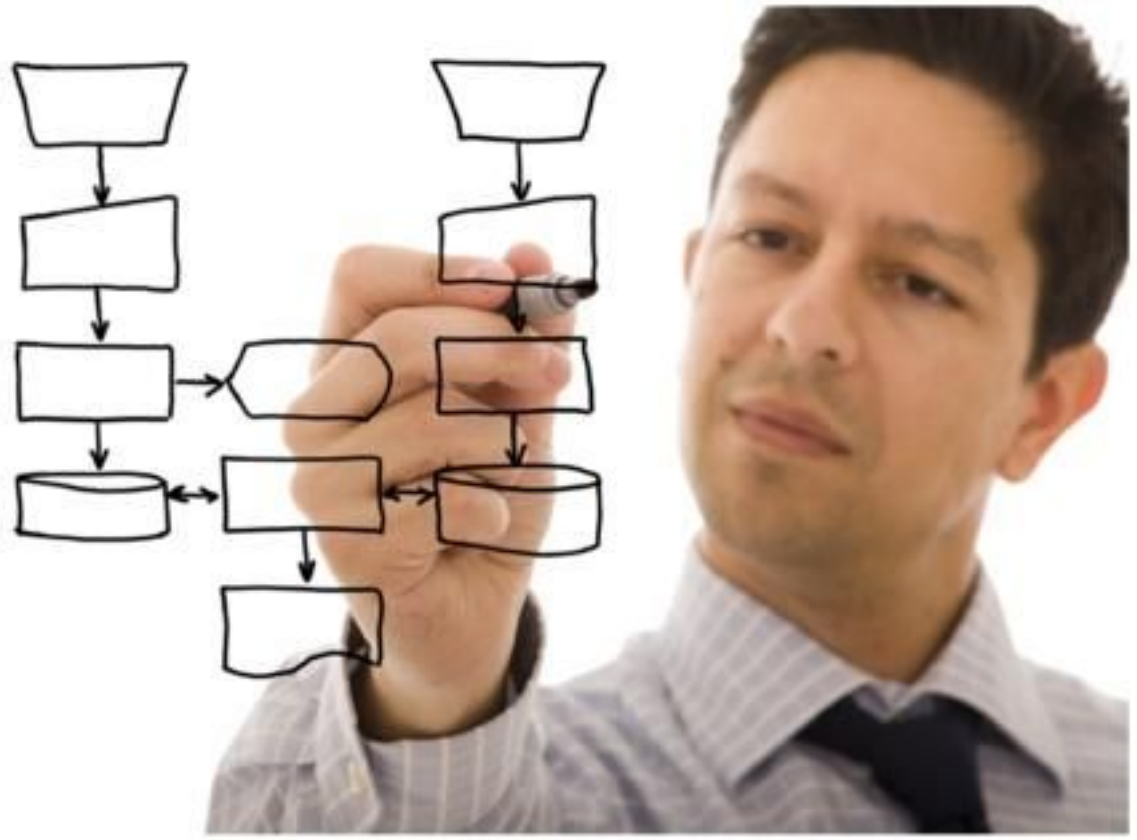


Ingeniería en DESARROLLO DE SOFTWARE

Los **programas que utilizan sólo estas tres instrucciones de control básicas** o sus variantes (como los bucles for, repeat o la instrucción condicional switch-case), pero no la instrucción goto, **se llaman estructurados.**

Programación estructurada (llamada también programación sin goto) que hasta la aparición de programación orientada a objetos **se convirtió en la forma de programar más extendida.**

A pesar que la Programación Orientada a Objetos (esta la abordaremos más adelante en otra unidad) se enfoca en la reducción de la cantidad de estructuras de control para reemplazarlas por otros elementos que hacen uso del concepto de polimorfismo. **Aun así los programadores todavía utilizan las estructuras de control (if, while, for, etc.) para implementar sus algoritmos** porque en muchos casos es la forma más natural de hacerlo.



Ingeniería en DESARROLLO DE SOFTWARE

Ventajas del Paradigma

1. Los programas son más fáciles de entender, pueden ser leídos de forma secuencial y no hay necesidad de tener que rastrear saltos de líneas (GOTO) dentro de los bloques de código para intentar entender la lógica interna.
2. La estructura de los programas es clara, puesto que las sentencias están más ligadas o relacionadas entre sí.
3. Se optimiza el esfuerzo en las fases de pruebas y depuración. El seguimiento de los fallos o errores del programa (debugging), y con él su detección y corrección, se facilita enormemente.

Ingeniería en DESARROLLO DE SOFTWARE

Ventajas del Paradigma

4. Se reducen los costos de mantenimiento. Análogamente a la depuración, durante la fase de mantenimiento, modificar o extender los programas resulta más fácil.
5. Los programas son más sencillos y más rápidos de confeccionar.
6. Se incrementa el rendimiento de los programadores.