ESPECIFICACIÓN DE PROBLEMAS

Yolanda Ortega Mallén

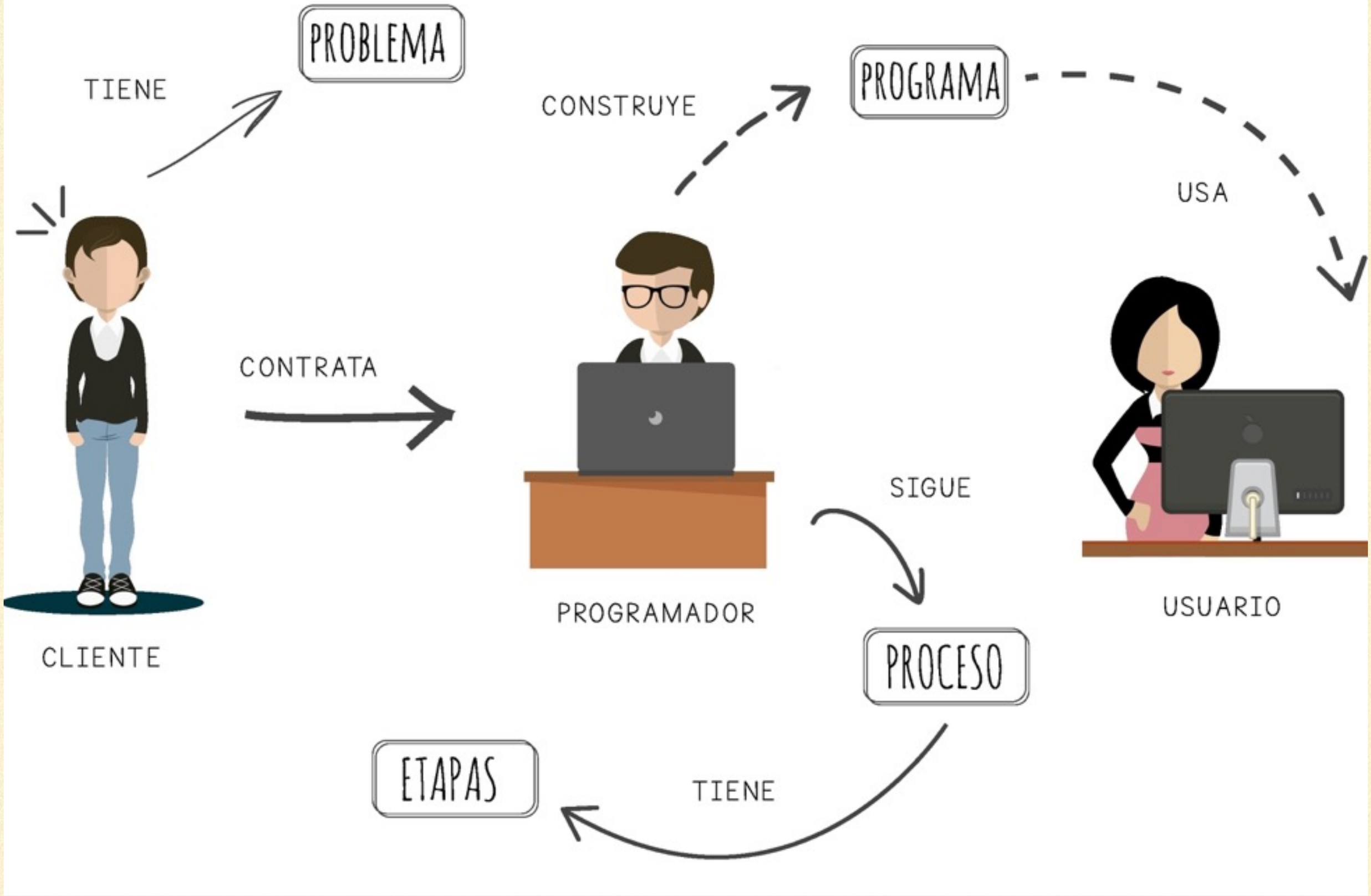
Sistemas Informáticos y Computación

SUMARIO

- Ciclo de vida del Software
- ¿Qué es una especificación?
- Especificaciones formales: técnica pre/post

BIBLIOGRAFÍA

- R.Peña. *Diseño de programas. Formalismo y abstracción.* 3ªedición. Prentice Hall, 2005. Capítulo 2.
- K. Broda y otros. *Reasoned Programming*. Prentice Hall, 1994. Capítulos 1 y 3.


PROBLEMA
TIENE
CONSTRUYE
PROGRAMA
USA
CONTRATA
SIGUE
CLIENTE
PROGRAMADOR
USUARIO
PROCESO
ETAPAS
TIENE

CICLO DE VIDA DEL SOFTWARE

Enunciado

Análisis

Especificación

Diseño

Algoritmo

Codificación

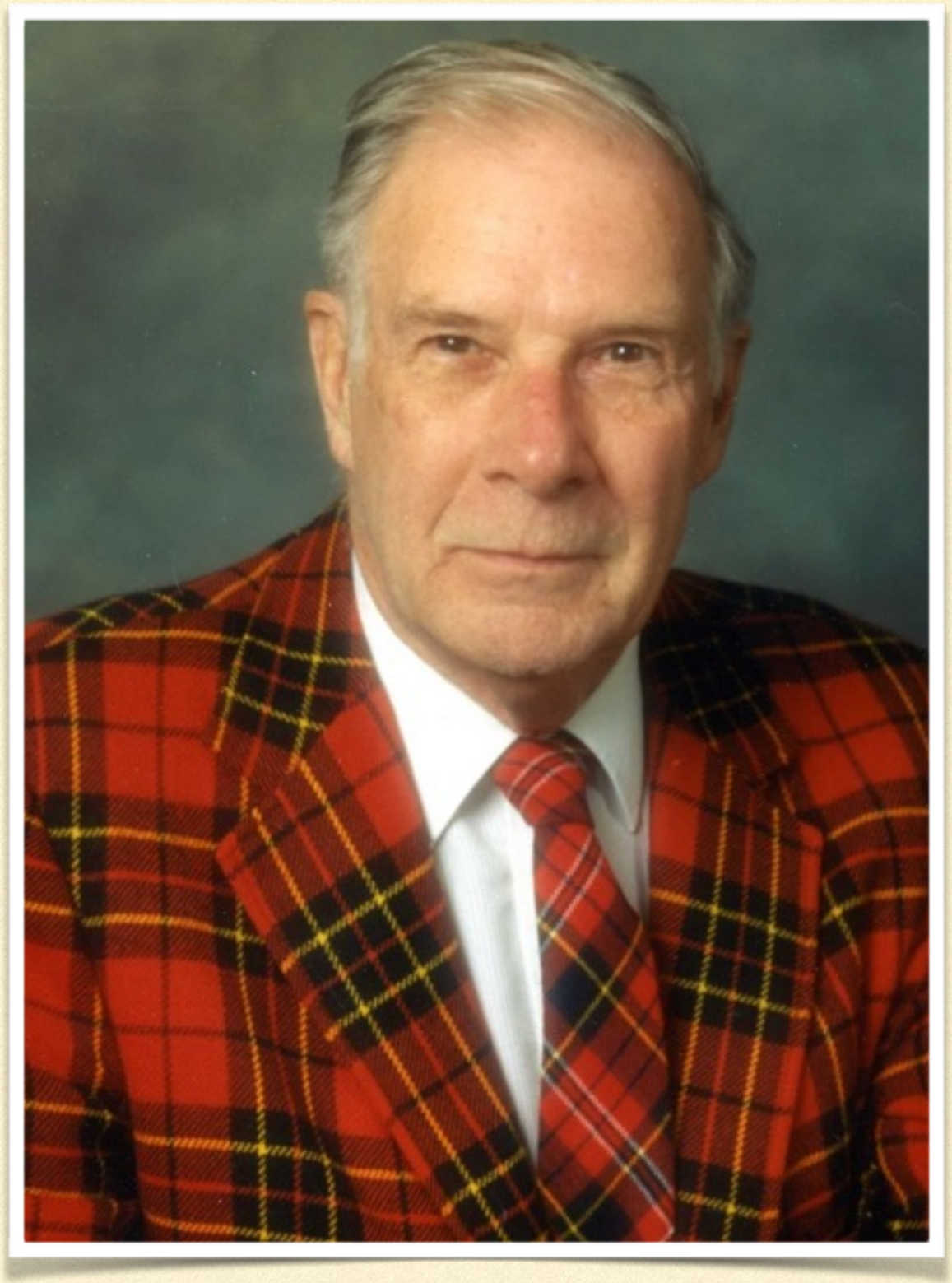
Programa

Mantenimiento

- Corrección de errores
- Modificación de requisitos

Teclear no es sustituto de pensar.

Richard W. Hamming

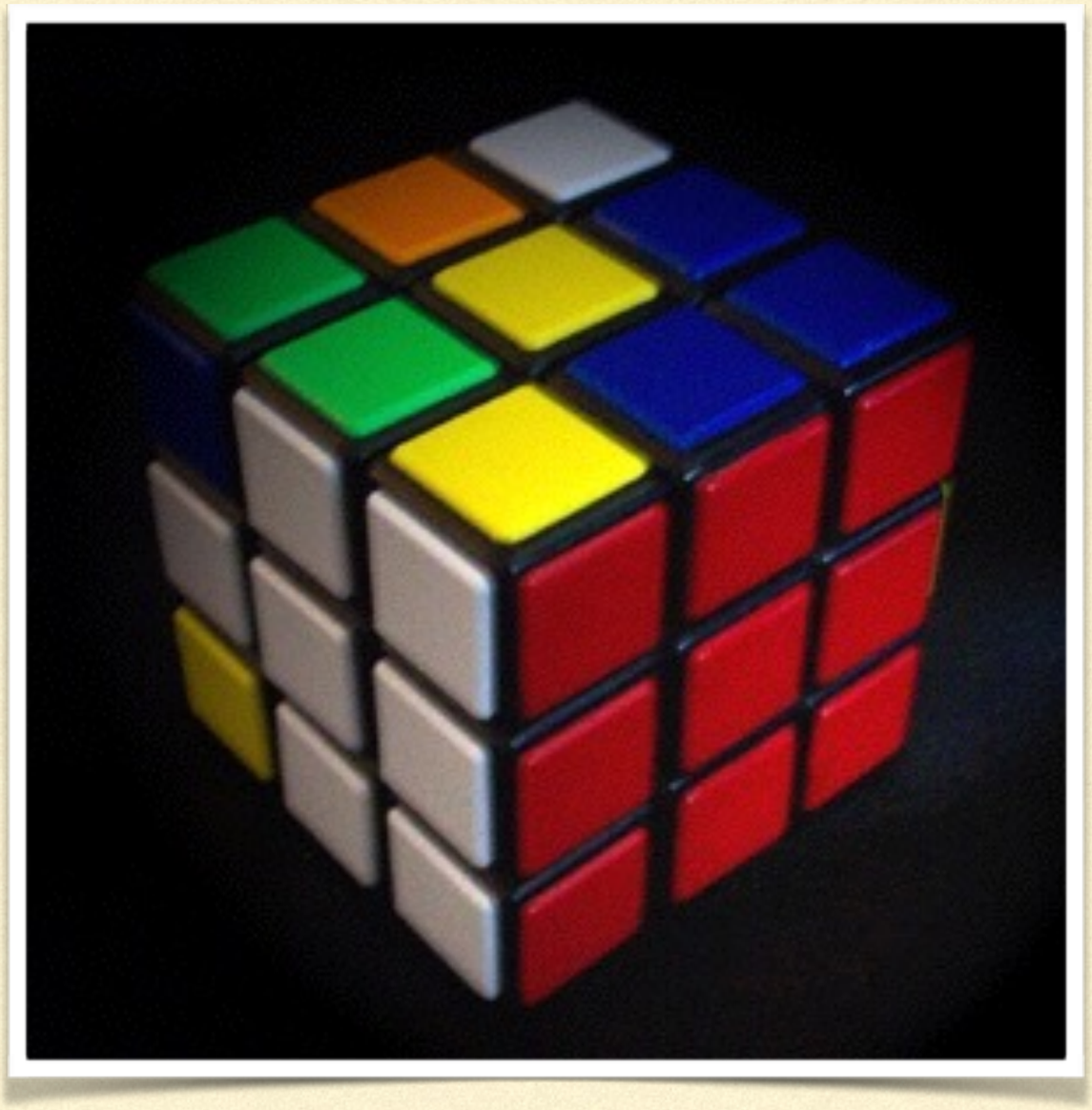


VALIDACIÓN (ESPECIFICACIÓN)

¿Resuelve el problema correcto?

VERIFICACIÓN (PROGRAMA)

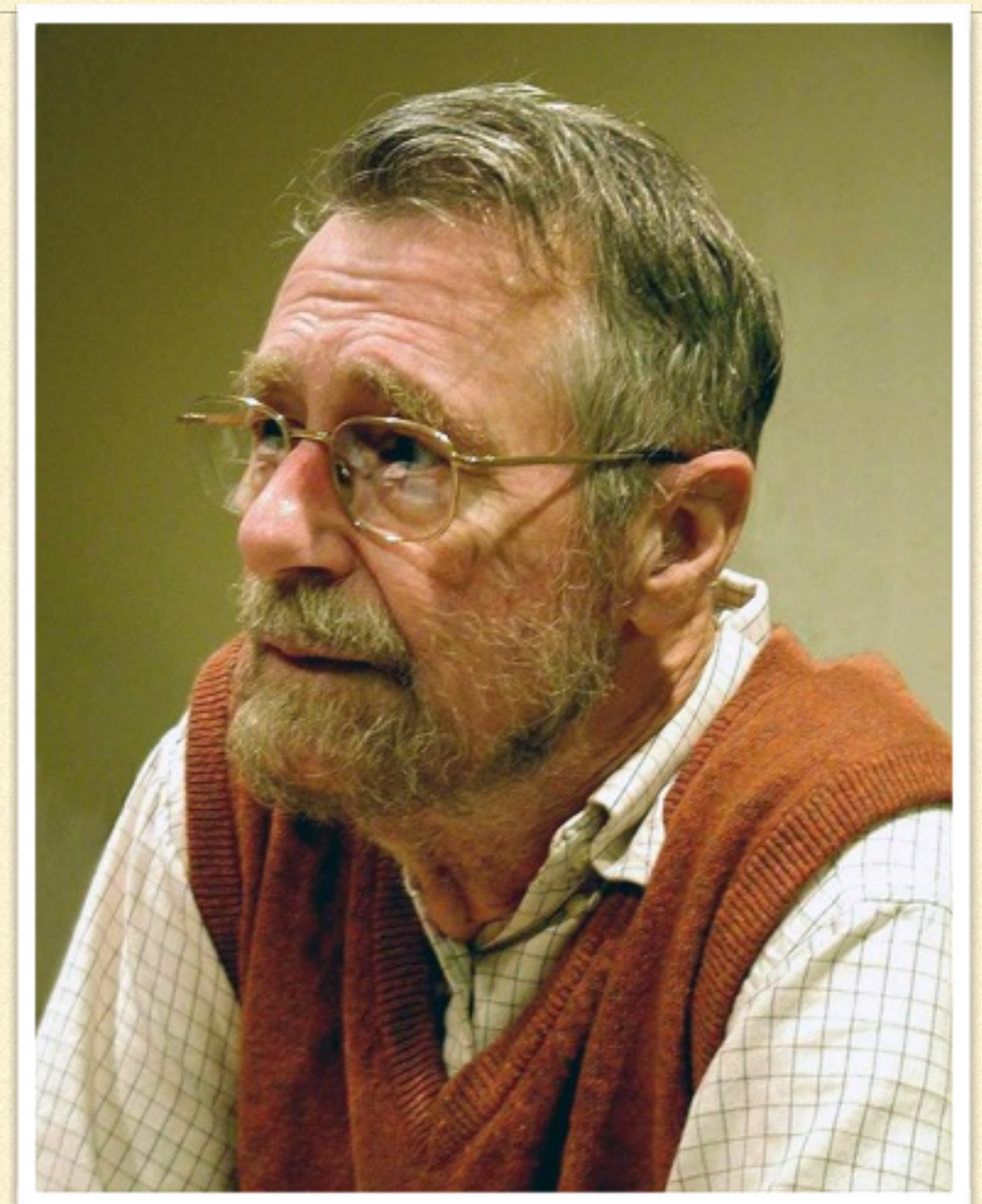
¿Resuelve correctamente el problema?



¿CORRECCIÓN DE PROGRAMAS?

La comprobación (*testing*)
solo puede establecer la
presencia de errores, nunca su
ausencia.

Edsger W. Dijkstra



COMPARAR DOS CADENAS

```
igual := (cadena1.long = cadena2.long)
si igual entonces
    para i := 1 hasta cadena1.long hacer
        igual := (cadena1.car[i] = cadena2.car[i])
```

Tests: pares de cadenas idénticas, de distinta longitud y cadenas diferentes pero de igual longitud.

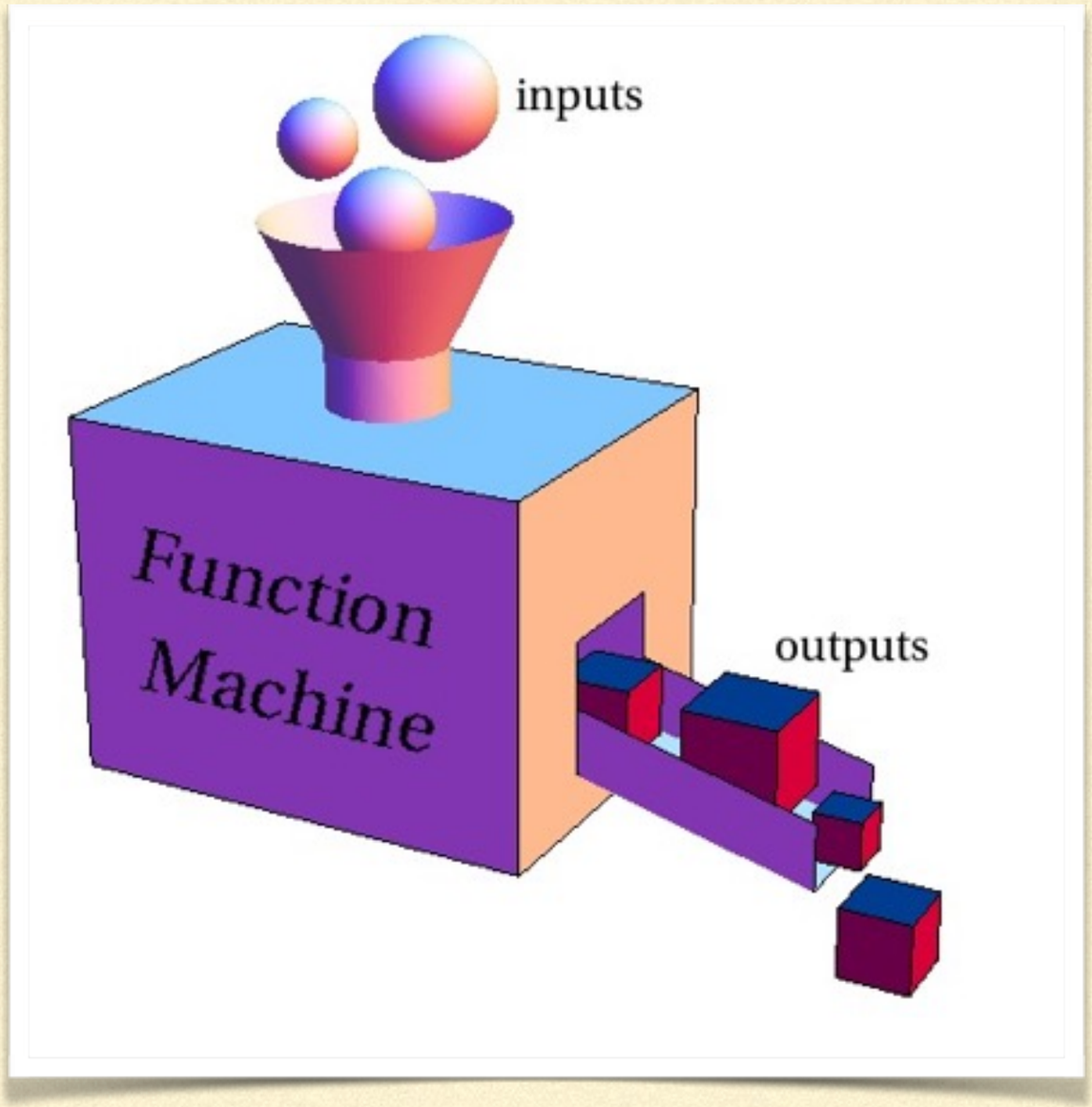
Probabilidad de encontrar un error:

10 tests – 33%

20 tests – 50%

> 50 tests – 90%

ABSTRACCIÓN FUNCIONAL



- Usuario: aspectos externos
- Programador: aspectos internos

¿QUÉ ES UNA ESPECIFICACIÓN?

- Programador (a priori):
  - Descripción detallada del problema a resolver.
  - Sin indicaciones de cómo resolverlo.

- Usuario (a posteriori):
  - Descripción detallada de lo que hace el programa.
  - Sin indicaciones de cómo lo hace.

UNA ESPECIFICACIÓN ES UN CONTRATO

- Obligaciones del usuario: restricciones de los datos de entrada.
- Obligaciones del programador: requisitos a satisfacer por todo programa válido.

REQUISITOS DE UNA ESPECIFICACIÓN

- Claridad: fácil de entender.
- Concisión: sin redundancia.
- Precisión: sin ambigüedad.
- Restricción: dejar fuera implementaciones inaceptables.
- Generalidad: no exigir condiciones innecesarias

Describir lo imprescindible.
Abstraer detalles irrelevantes.

ESPECIFICACIONES FORMALES

- Concisión y precisión.
- Razonamiento formal para demostrar la corrección.
- Técnica pre/post: lógica de predicados.

Pre-condición: requisitos del estado inicial.
Post-condición: requisitos del estado final.

{A} P {B}

Si P comienza su ejecución en un estado que satisface A,
entonces P termina y alcanza un estado que satisface B.

EJEMPLO

- Enunciado: dada una secuencia de valores enteros y un entero n, indicar si el valor de alguno de los primeros n elementos de la secuencia equivale a la suma de todos sus precedentes.

¿Puede ser n negativo? ¿nulo?
¿mayor que el número de elementos?
¿Y si el primer elemento es nulo? ¿Devolver cierto o falso?

- Especificación:

$$\{(s = a_1\ a_2\ \ldots\ a_m) \wedge (m \geq 0) \wedge (0 \leq n \leq m)\}$$

fun es_suma (s : **sec de** ent; n : ent) **dev** b : bool

$$\{b = \exists\ i : 1 \leq i \leq n : a_i = \sum_{j=1}^{i-1} a_j\}$$

COMPROMISO

- Usuario:
  - Pre-condiciones débiles → datos de entrada más generales.
  - Post-condiciones fuertes → más resultados y más precisos.
- Programador:
  - Pre-condiciones fuertes → menos casos.
  - Post-condiciones débiles → resultados más generales.

DIVISIÓN ENTERA

{b ≠ 0}

fun divide (a, b : nat) **dev** c, r : nat

{a = b * c + r} (r ≥ 0) ∧ (r < b)

Post-condición demasiado débil:

c := 0; r := a