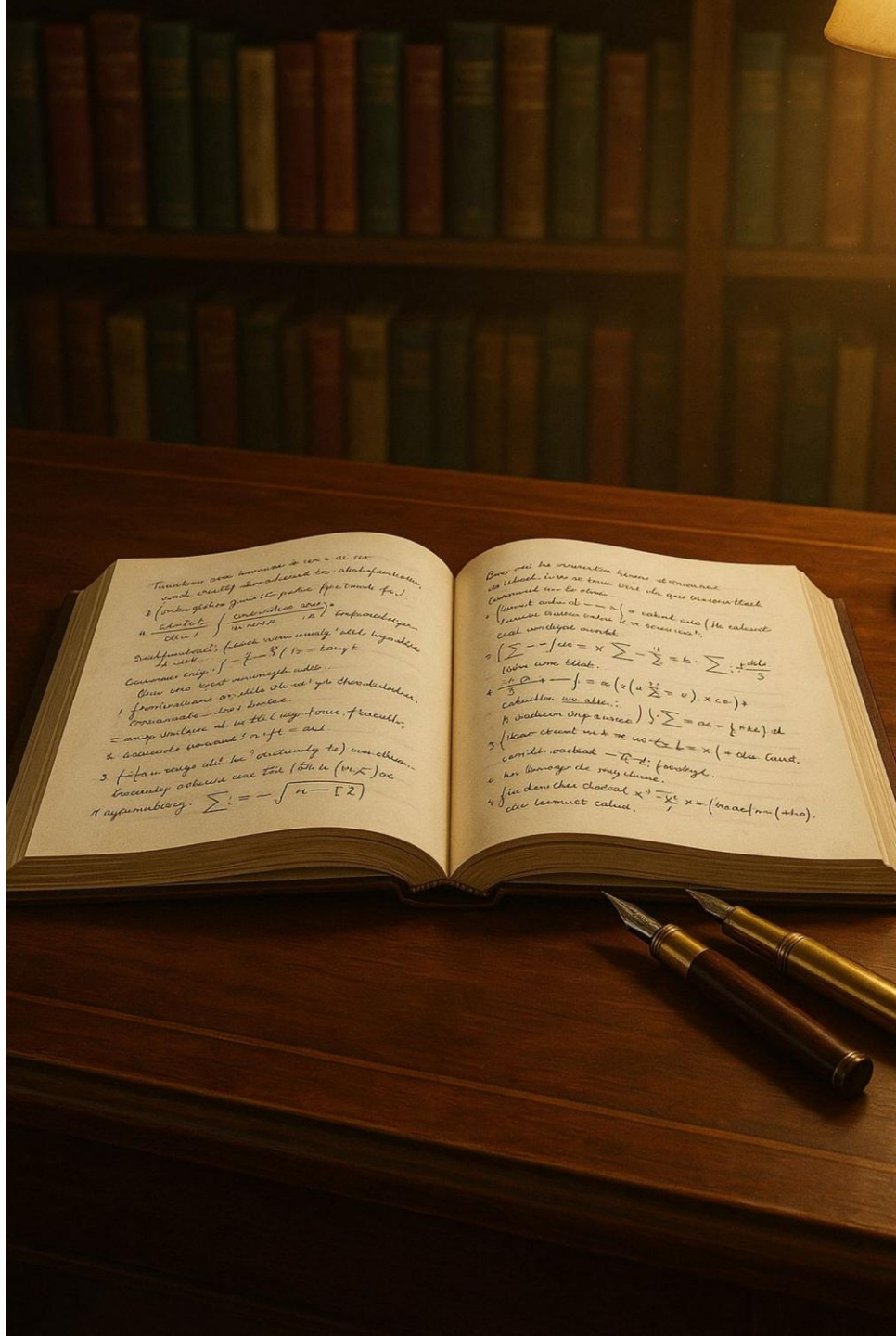
Verificación de Funciones y Algoritmos

Fundamentos de Verificación

DESARROLLO FORMAL DE SOFTWARE

INGENIERÍA DE SISTEMAS · UNIVERSIDAD DEL PACÍFICO



1

Verificación de funciones

Analizar un método mediante su **contrato**: PRE y POST.

2

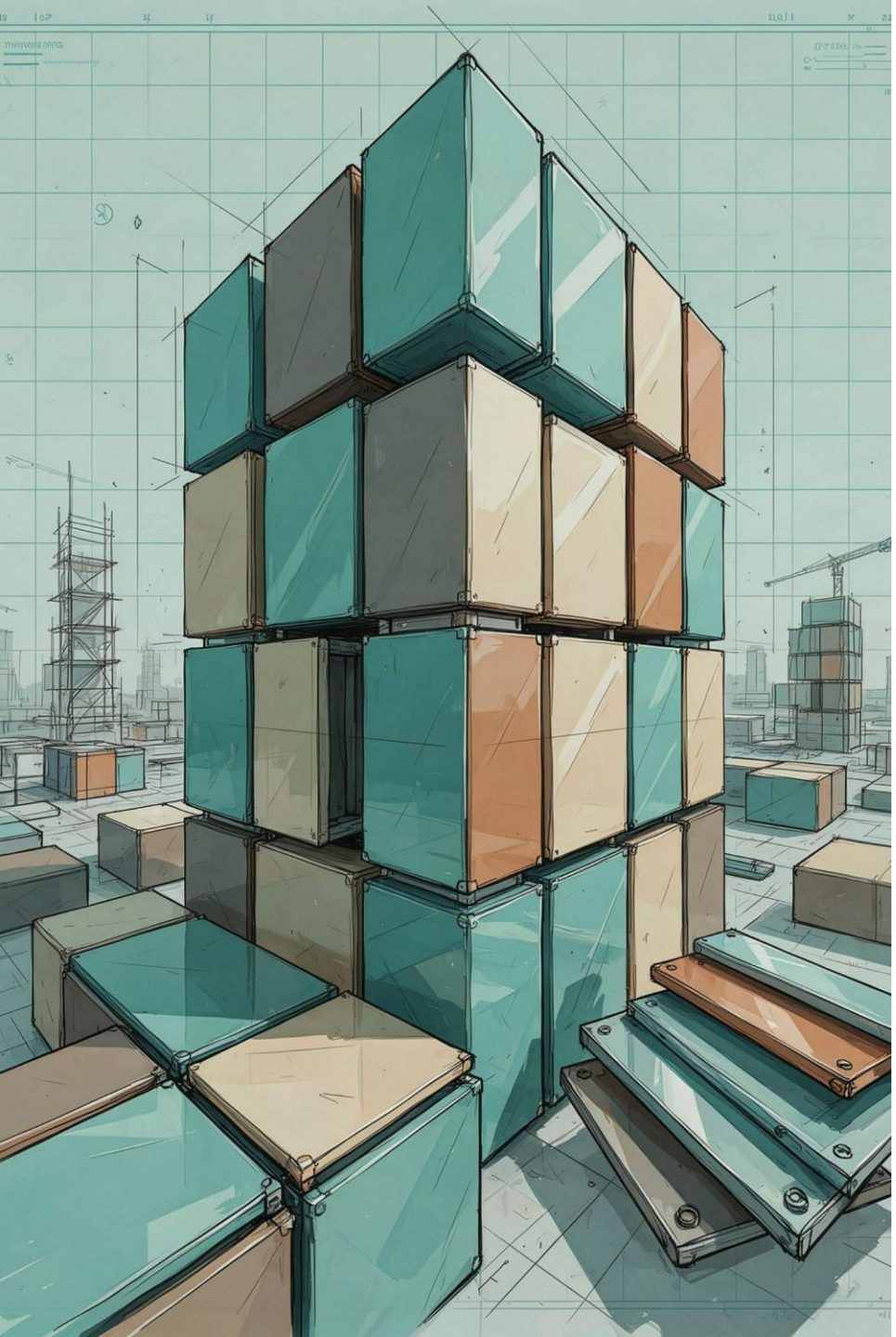
Verificación de algoritmos

Justificar que los pasos producen el **resultado especificado**.

3

Relación PRE–ejecución–POST

En ambos casos conectamos las condiciones de entrada, ejecución y salida.



Verificación modular de funciones

PRE — Precondición

Condiciones que debe respetar **quien llama** al método antes de invocarlo.

POST — Postcondición

Propiedad que debe garantizar **quien implementa** si se cumple PRE y la ejecución termina.

Modularidad

La implementación puede cambiar internamente siempre que conserve el contrato. Facilita pruebas, mantenimiento y localización de defectos.

by ykcaicedo

Ejemplo: `calcularCosto()`

PRE

`horas > 0` y `tarifa >= 0`

Lógica

El método valida los datos y devuelve `horas × tarifa`

POST

`resultado = horas × tarifa`

Traza concreta

Con `horas = 3` y `tarifa = 12 000`:

> **resultado = 36 000**

Si se incumple la PRE, el método rechaza la entrada conforme al diseño del contrato. El comportamiento fuera de PRE **no está garantizado**.

¿Qué debe saber quien llama?



Argumentos

Qué valores concretos proporcionar al invocar el método.



Precondiciones

Qué condiciones debe garantizar **antes** de realizar la llamada.



Garantía de resultado

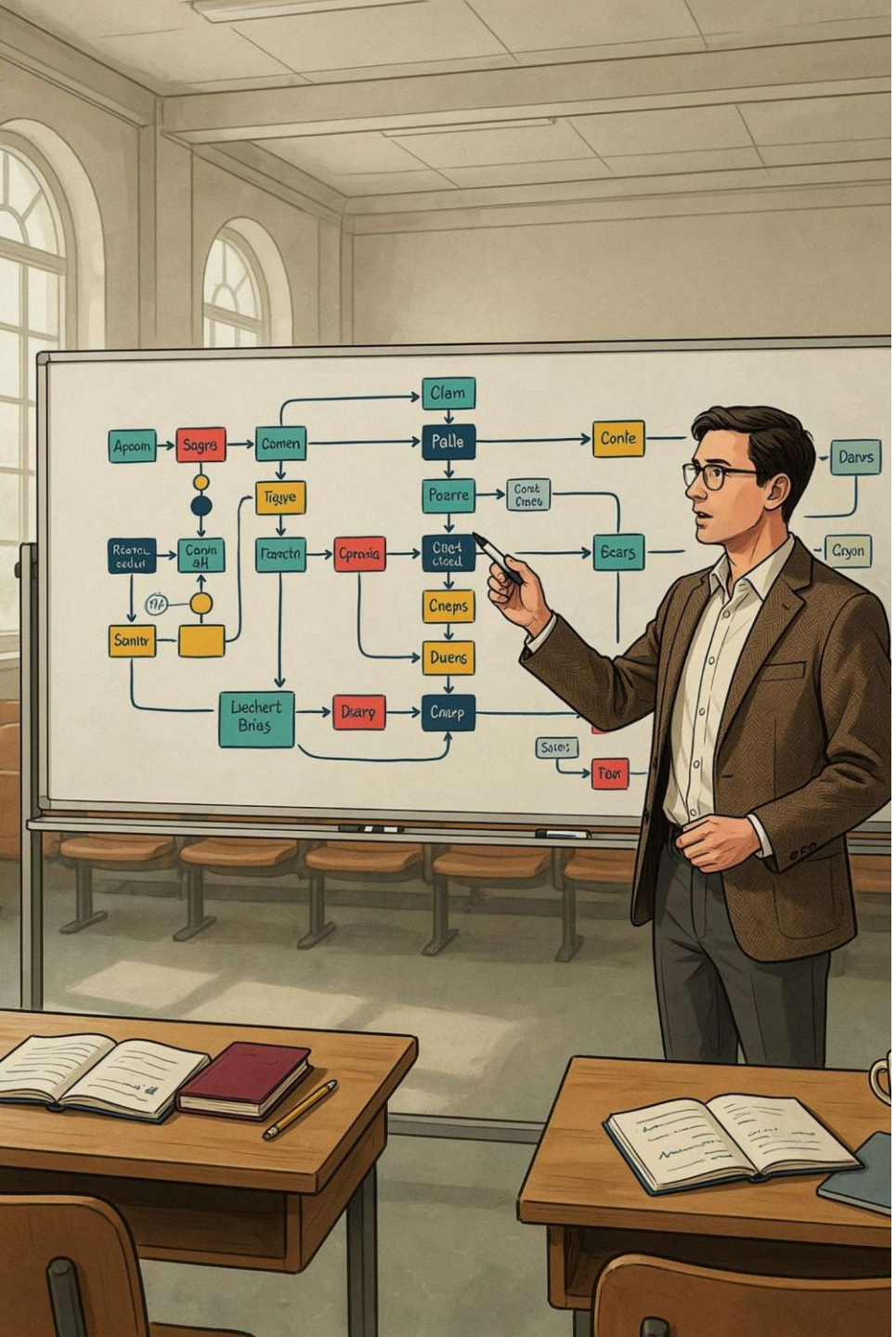
Qué propiedad ofrece el método una vez que termina correctamente.



Abstracción interna

No es necesario conocer cada detalle de la implementación si el contrato está bien definido.

by ykcaicedo



Verificación de algoritmos

01

Definir entradas

Establecer el conjunto de entradas válidas que el algoritmo puede recibir.

02

Especificar resultado esperado

Indicar con precisión qué propiedad debe cumplir la salida.

03

Analizar pasos y caminos

Recorrer los pasos del algoritmo y todos los flujos de control posibles.

04

Revisar casos relevantes

Comprobar casos normales, límites y situaciones especiales.

Ejemplo: máximo de dos números



Especificación

Entrada: dos números `a` y `b`.

POST: `máximo >= a`, `máximo >= b` y `máximo` coincide con `a` o con `b`.

Si `a = b`, cualquiera de los dos representa el máximo; ambas ramas son correctas.

by ykcaicedo

Razonar por casos

Caso `a >= b`

Se elige `a`. Por hipótesis `a >= b`, y `a = a`, de modo que la POST se satisface con `máximo = a`.

Caso `a < b`

Se elige `b`. Sabemos que `b > a`, y `b = b`, de modo que la POST se satisface con `máximo = b`.

Ambas ramas cubren todas las entradas posibles y justifican que el resultado es el máximo.

by ykcaicedo

Función dentro de un algoritmo



Composición modular

Un algoritmo puede invocar `calcularCosto()` como operación auxiliar. Para ello:

- El algoritmo debe garantizar la **PRE** de la función antes de llamarla.
- A cambio, puede asumir la **POST** de la función en los pasos siguientes.
- Los contratos deben ser **coherentes** con el flujo de control general.

by ykcaicedo

Pruebas vs. Verificación

1 Prueba

Ejecuta un **caso concreto** con valores específicos y comprueba el resultado.

Detecta defectos puntuales, pero no puede cubrir todos los casos posibles.

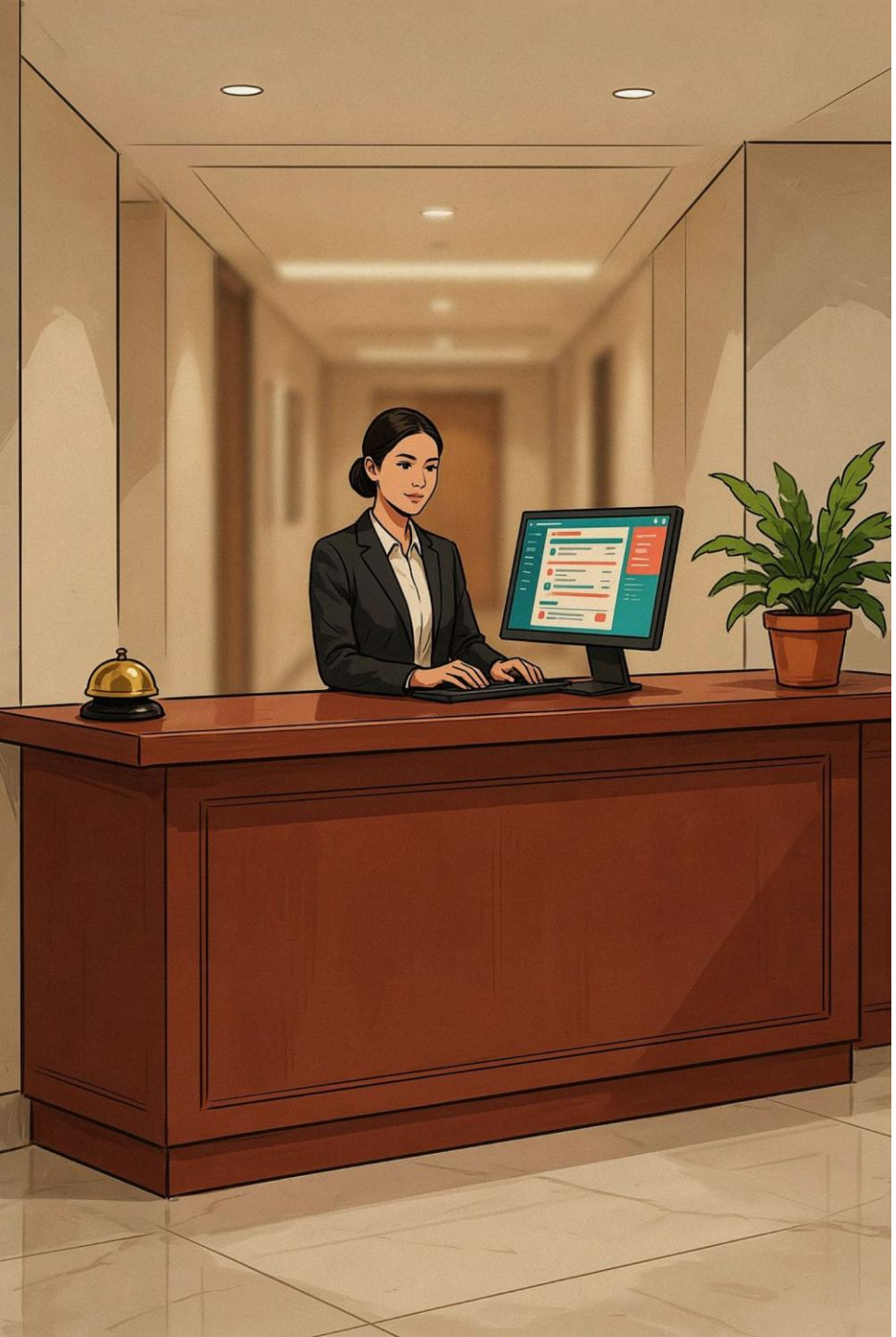
2 Verificación

Razona sobre **todas las entradas** que cumplen la especificación, con argumentos lógicos.

Proporciona una justificación general que ninguna prueba aislada puede reemplazar.

Las pruebas y la verificación son complementarias: una detecta errores concretos; la otra los descarta en general.

by ykcaicedo



Aplicación: sistema de reservas

`consultarDisponibilidad()`

Define con precisión qué significa que haya cupos libres en el sistema.

`calcularCosto()`

Establece las entradas válidas y la regla de cálculo del precio de la reserva.

`registrarReserva()`

Coordina ambas operaciones y valida que se cumplan todas las condiciones previas.

Cada método y el algoritmo general deben tener **contratos coherentes entre sí**.

by ykcaicedo

Errores frecuentes

PRE o POST vagas

Condiciones imprecisas impiden razonar formalmente sobre el comportamiento del método.

Llamar sin respetar PRE

Invocar un método sin garantizar su precondición invalida toda garantía sobre el resultado.

Probar solo un ejemplo

Un único caso numérico no reemplaza la justificación general que exige la verificación.

Ignorar una rama

No analizar ambas ramas de un condicional deja el algoritmo parcialmente verificado.

by ykcaicedo

Reto oral

1

PRE incumplida

Si `horas = 0` incumple la PRE, ¿qué debe ocurrir y por qué el comportamiento no está garantizado?

2

Perspectiva del llamador

¿Qué información necesita quien invoca `calcularCosto()` para usarlo correctamente?

3

Análisis de casos

¿Por qué es imprescindible analizar **los dos casos** del algoritmo del máximo para completar la verificación?



Cierre

Verificación de funciones

Una función se verifica mediante su **contrato**: PRE define las obligaciones del llamador y POST las garantías del implementador.

Verificación de algoritmos

Un algoritmo se verifica justificando, caso a caso, que sus pasos producen el **resultado especificado**.

Modularidad

La modularidad permite **combinar ambos razonamientos**: los contratos de las funciones se integran en la justificación del algoritmo general.

by ykcaicedo