



Prueba de Software

¿Qué son las pruebas?

A dark blue background with faint, light blue illustrations. On the right, a pen is shown diagonally. In the center and bottom right, there are checkmarks inside rounded rectangular boxes. A dashed line is visible in the bottom left corner.



*“Testing shows the presence, not
the absence of bugs”*

*“Las pruebas muestran la
presencia, no la ausencia de
errores.”*

Edsger Dijkstra

Sobre las pruebas



La prueba consiste en la búsqueda de errores en el sistema.

Que las pruebas no pongan en evidencia los errores, no significa que el sistema no los tenga.

Un sistema mal probado al cual no se le detectaron errores, no significa que no los tenga. (parece obvio ¿no?)

Especificidad de la prueba



Una buena prueba es aquella que se enfoca en un aspecto específico y puntual del sistema, ya que al fallar, permite identificar fácilmente dónde está el error.

Una prueba muy abarcativa que cubre múltiples aspectos en simultáneo del sistema, no suele ser de gran utilidad, ya que al fallar, no brinda información clara de cuál puede ser el error, solo nos hace evidente la existencia del mismo.

Pruebas en el ciclo de vida del software

A dark blue background with a faint, stylized illustration of a pen writing checkmarks on a document. The pen is positioned diagonally in the upper right, and two large checkmarks are visible on the document below it.

Prueba en el ciclo de vida del software



Antes de entender cómo impactan las pruebas en el ciclo de vida del software, vamos a realizarnos la siguiente pregunta:

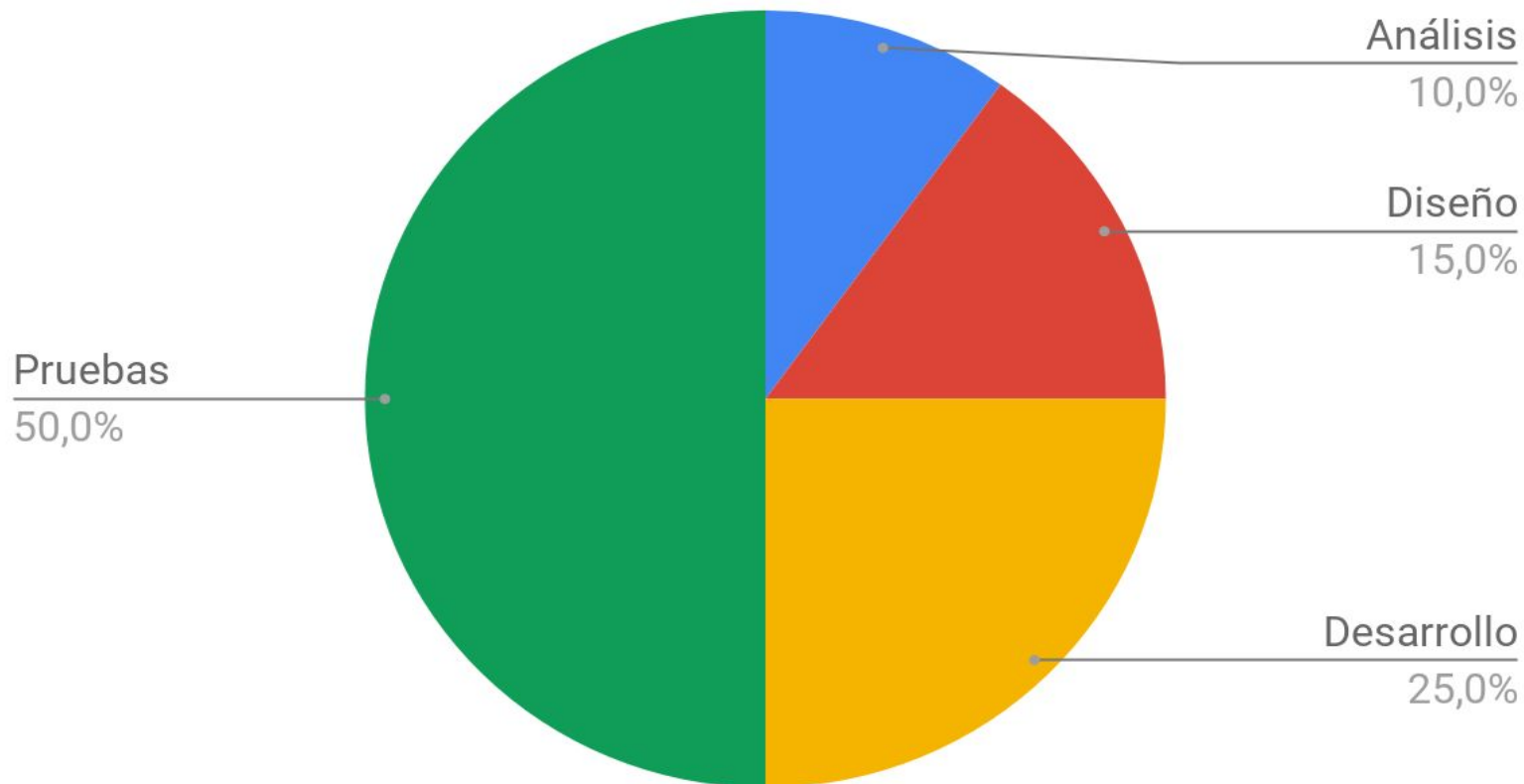
¿Qué porcentaje del tiempo de nuestro proyecto dedicamos a cada etapa del mismo?

Análisis - Diseño - Desarrollo - Pruebas

Tiempo empleado en cada etapa (estimativo)



Ciclo de vida del proyecto



Etapas de trabajo para el cursado de la materia



A pesar de que el gráfico anterior muestra que las pruebas componen el 50% del tiempo de vida del proyecto, no significa que necesariamente este tiempo deba estar dado sobre el final del mismo.

De hecho, durante el cursado de esta materia utilizaremos (obligatoriamente) una metodología para realizar las pruebas que establece que las mismas deben realizarse ANTES de la etapa de desarrollo.

La pregunta es...



¿Cómo pruebo algo que aún no está desarrollado?



La prueba primero



Para responder a la pregunta, dividiremos la etapa de pruebas en 2 sub-etapas:

- Primero, nos encargaremos de la **preparación de la prueba** y se realizará luego del análisis y antes del diseño y desarrollo de la solución.
- Al finalizar el desarrollo (o incluso mediante), se realizará la **ejecución de la prueba**, en donde aplicaremos todas las pruebas que planificamos anteriormente.

Etapas de trabajo para el cursado de la materia



1. Análisis
2. Preparación de la prueba
3. Diseño
4. Desarrollo
5. Ejecución de la prueba

Notar que la etapa de preparación de la prueba es previa al diseño y desarrollo.

¿Por qué?

La prueba primero



Al realizar la preparación de la prueba a continuación del análisis, podremos entender y comprender en detalle todos los requerimientos del problema (o proyecto).

Al analizar por ejemplo las salidas que esperamos para cada entrada diferente de nuestro sistema, podemos descubrir y profundizar sobre ciertos casos particulares o casos borde que tiene el problema.

La prueba primero



Descubrir los casos particulares y cualquier otro detalle relevante del problema antes de las etapas de diseño y desarrollo, nos conducirá hacia una **solución de calidad**.

Durante la etapa de diseño, podremos tener en cuenta todos estos detalles para diseñar una solución eficaz, eficiente y elegante.

Llegaremos a la etapa de desarrollo con un problema bien analizado y un diseño sólido.

La prueba primero



Finalmente, una vez que tengamos el desarrollo “finalizado” (o por lo menos así lo creamos), llega el momento de **ejecutar la prueba**.

La ventaja que tenemos es que nuestra prueba ya está planificada, por lo que la ejecución se realiza de forma simple, ordenada y principalmente en forma rápida.

En caso de detectar algún error, tenemos nuestro desarrollo recientemente terminado, por lo que hacer cambios nos será mucho más fácil.

Métodos de prueba de software

A dark blue background with a faint, stylized illustration of a pen and two checkmarks. The pen is positioned diagonally in the upper right, and the checkmarks are located in the center and lower right.

Pruebas de caja negra

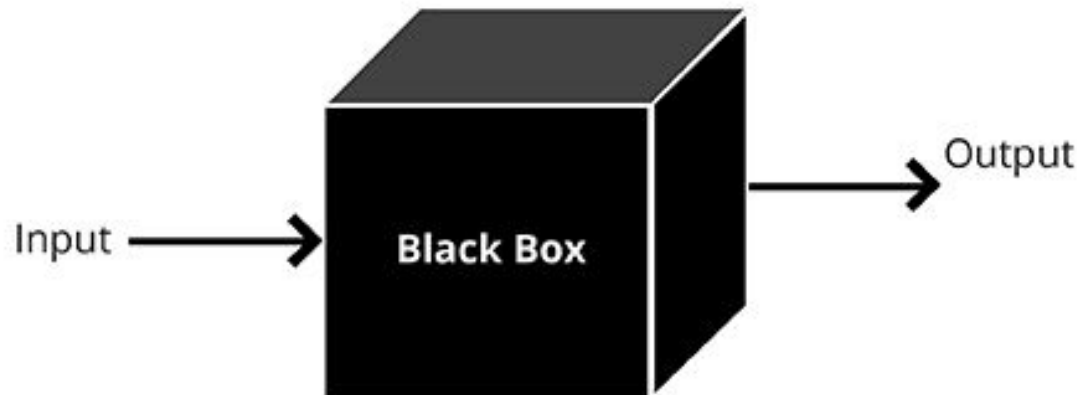


En este tipo de pruebas, se ve al sistema como una caja negra, haciendo referencia a que se ve solo la parte exterior de la caja y no es posible ver el interior.

Las pruebas de caja negra, se basan en la premisa de que quién ejerce el rol de ejecutor de la prueba o tester, no tiene acceso a los detalles de implementación del mismo (código, estructuras utilizadas, algoritmos utilizados, patrones de diseños utilizados, etc.).

Pruebas de caja negra

BLACK BOX TESTING APPROACH



Buscan verificar que la relación entre las entradas y las salidas sea correcta.

Pruebas de caja blanca (o caja de cristal)

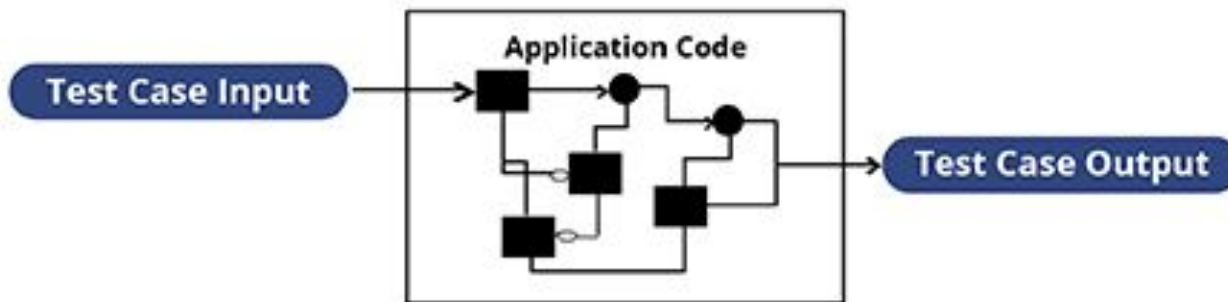


A diferencia de las pruebas de caja negra, las pruebas de caja blanca o de cristal, hacen referencia a que además del exterior, se puede ver hacia el interior de la caja.

Las pruebas de caja blanca, se basan en la premisa de que quién ejerce el rol de ejecutor de la prueba o tester, tiene acceso a cada detalle de implementación del mismo. Puede acceder al código, saber qué estructuras de datos y algoritmos fueron utilizados para la construcción del sistema, entre otras cosas.


Pruebas de caja blanca (o caja de cristal)

WHITE BOX TESTING APPROACH




Buscan errores en el código fuente, además de verificar el comportamiento del usuario con la interfaz.

Métodos de caja negra

- 
- Pruebas funcionales
 - Pruebas de aceptación
 - Pruebas alfa
 - Pruebas beta
 - Lote de prueba
 - Programa probador
 - Etc.

Métodos de caja blanca

- 
- Inspección de código
 - Pruebas unitarias
 - Pruebas de integración
 - Pruebas de estrés
 - Prueba de caminos básicos
 - Etc.

Métodos de prueba a utilizar en la materia



A pesar de que cualquier método que los alumnos creen necesario para probar y asegurar la calidad del desarrollo es válida, durante el cursado de la materia nos enfocaremos en los siguientes 2 métodos:

- Lote de prueba
- Programa probador

Lote de prueba

A dark blue background featuring faint, light blue illustrations. On the right side, there is a detailed drawing of a pen. Scattered across the background are several checkmarks, some enclosed in rounded rectangular boxes, suggesting a checklist or a list of items being reviewed.

Lote de prueba



El lote de prueba es una técnica de prueba de caja negra en la que dada una **entrada**, busca validar que la **salida del sistema** se corresponda con la **salida esperada**.

El lote de prueba define múltiples casos de prueba.

Cada caso de prueba debe definir:

- Nombre del caso (debe ser significativo)
- Descripción (qué error busca)
- Entrada
- Salida esperada

Ejemplo de definición de caso de prueba

Nombre: caso_01_no_considera_al_ultimo_usuario

Descripción: Busca detectar un error en caso de que no se procese al último usuario de la lista.

IN	OUT (esperado)
5 2 5 3 2 7 7 1 6 2 7 3 3 7 2 6	5 15

Ejemplo de definición de caso de prueba

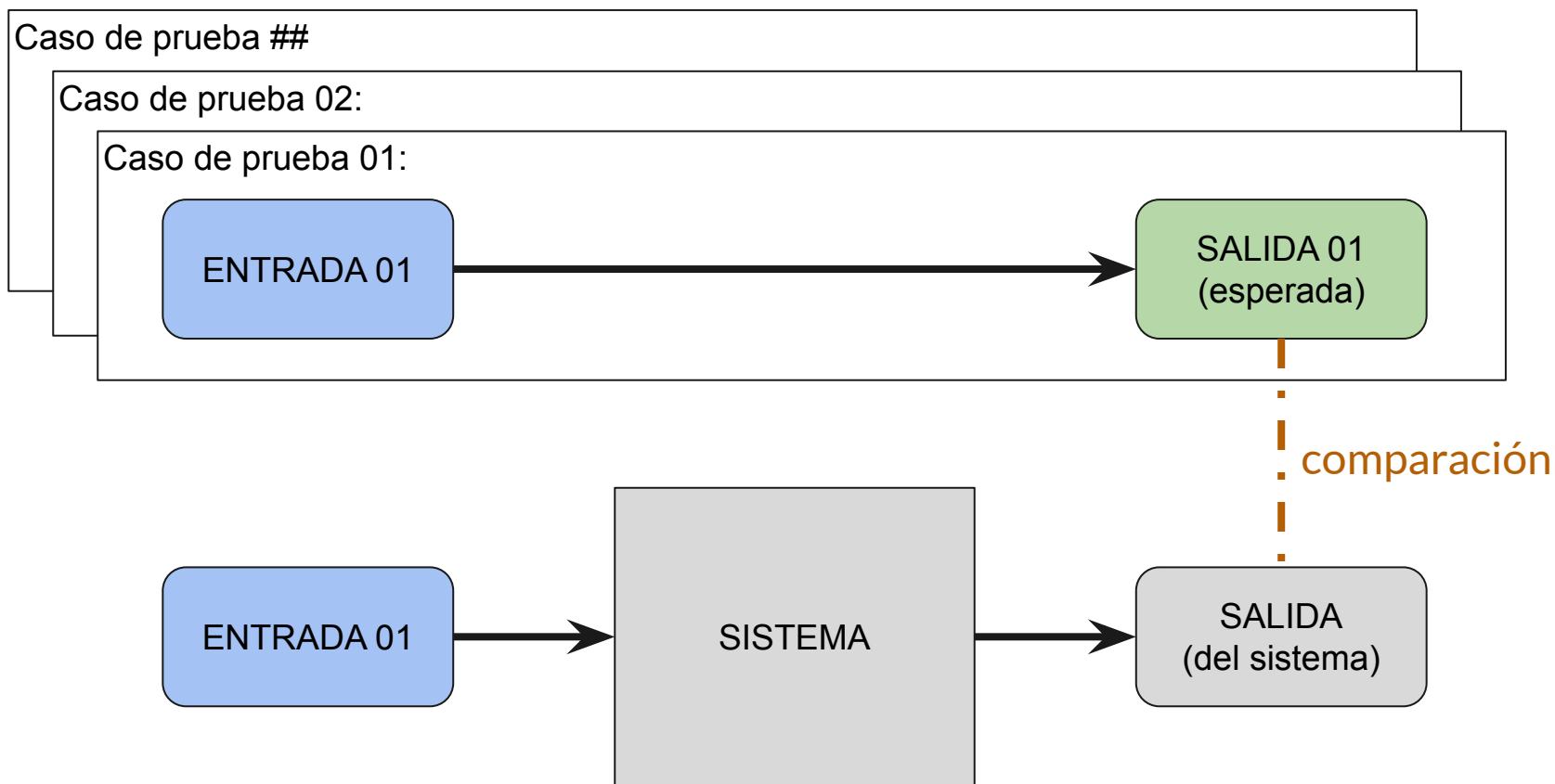
Nombre: caso_01_no_considera_al_ultimo_usuario

Descripción: Busca detectar un error en caso de que no se procese al último usuario de la lista.

IN	OUT (esperado)
5 2 5 3 2 7 7 1 6 2 7 3 3 7 2 6	5 15

Centrarse en cómo se debe definir un caso de prueba y no en el caso en sí, ni qué problema resuelve.

Lote de prueba



Casos de prueba



Los casos de prueba, pueden contemplar:

- Casos de enunciado del problema
- Casos de flujos normales
- Casos de flujos particulares y/o excepcionales
- Casos borde (extremos, tipo de dato, etc.)
- Casos de fatiga
- Etc.

Lote de Pruebas - Salidas múltiples



¿Qué sucede si el problema tiene múltiples salidas válidas para una determinada entrada?

En ese caso, un caso de prueba parece no ser la mejor forma de realizar la prueba.

Si solo indicamos una de todas las posibles salidas válidas, en caso de que la persona que ejecute la prueba no lo sepa, podría comparar las salidas y que las mismas no coincidan (aunque podría ser válida también).

Lote de Pruebas - Salidas múltiples



Tampoco podríamos listar todas las posibles salidas, porque podrían ser demasiadas (o incluso infinitas).

En resumen, si un problema tiene salidas múltiples:

- El lote de prueba no es un método apropiado para los casos con salidas múltiples.
- Se puede solo probar casos con salidas únicas (¿pero cómo garantizamos la calidad de los demás casos?).
- Si las salidas son múltiples pero finitas (NO más de 3 o 4 posibilidades), se podrían listar TODAS las posibilidades y hacer una aclaración al respecto.

Casos de prueba



Recordar:

- Un buen caso de prueba, es aquel que se enfoca en un único aspecto del sistema, tal que si falla, permite detectar fácilmente cuál es el error.
- No hay una cantidad mínima de casos de prueba que haya que hacer, la cantidad depende de cada problema y de la forma y calidad con la que estén pensados los casos.

Casos de prueba: ¿Cómo identificarlos?

- Cada parte del problema que genere dudas durante la lectura y luego se termine entendiendo qué es realmente lo que se pide, amerita un caso de prueba.
- Una buena forma de detectar casos de prueba, es que dos o más personas analicen un mismo problema y luego lo pongan en común. Las partes que no hayan quedado claras o en las que haya diferencia entre lo que cada uno entendió, seguramente ameriten uno o más casos para clarificar las mismas.

Programa probador

The background is a solid dark blue. Overlaid on this background are faint, light blue illustrations. On the right side, there is a detailed illustration of a pen, angled diagonally. Scattered across the left and center are several checkmarks. Some checkmarks are contained within rounded rectangular boxes, while others are standalone. The overall aesthetic is clean and professional, suggesting a theme of testing or approval.

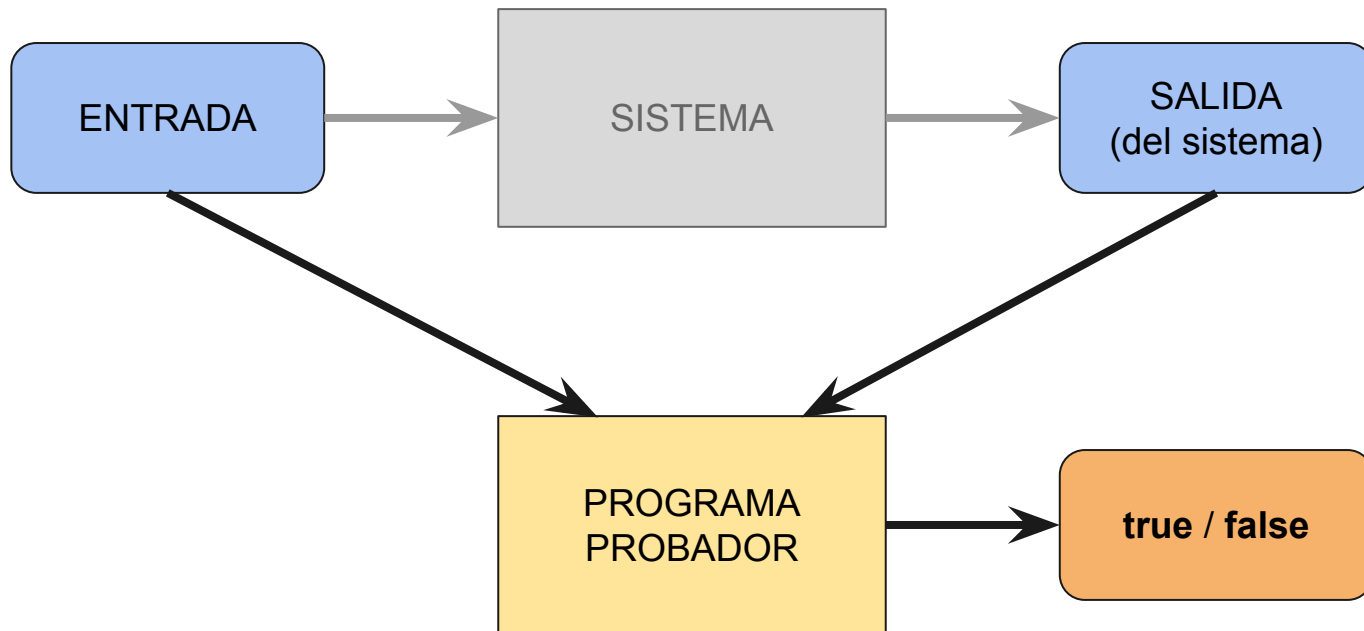
Programa Probador



Un programa probador, consiste en un programa independiente del sistema que resuelve el problema original, que tiene como fin, verificar que una salida obtenida por el sistema se corresponda con la entrada recibida.

El programa probador, recibe tanto la entrada como la salida del sistema y el resultado es un valor de verdad (true/false) que indica si el resultado es válido o no.

Programa probador



Programa probador: complejidad



En resumen, el programa probador debería recibir tanto la entrada como la salida que arrojó el sistema, y verificar que la misma se corresponda, pero... verificar que la salida se corresponda con la entrada, ¿no tiene la misma complejidad que el sistema original?

En un primer momento, podríamos pensar que el programa probador debería resolver el mismo problema que el sistema original, por lo tanto la complejidad sería la misma y el riesgo de introducir un error en el código se mantiene.

Programa probador: complejidad



Sin embargo, un punto importante es que SIEMPRE la complejidad del programa probador debe ser significativamente inferior a la del sistema original.

Por lo tanto, el programa probador NO DEBE nunca resolver el problema original, solo debe validar que la salida arrojada sea correcta.

Para ciertos problemas, este sub-problema de validación es significativamente inferior en complejidad que el problema original.

Programa probador: complejidad - ejemplo 1

Supongamos un problema simple, realizar una división (clásica) entre 5832 dividido 6:

$$\begin{array}{r} 5833 \quad | \quad 6 \\ -54 \quad \quad 972 \\ \hline 433 \\ -42 \\ \hline 13 \\ -12 \\ \hline 1 \end{array}$$

El input del problema original es:

- el dividendo: 5833
- el divisor: 6

El output del problema es:

- el cociente: 972
- el resto: 1

Programa probador: complejidad - ejemplo 1

Supongamos un problema simple, realizar una división (clásica) entre 5832 dividido 6:

$$\begin{array}{r} \text{5833} \mid \text{6} \\ \underline{-54} \quad \text{972} \\ 433 \\ \underline{-42} \\ 13 \\ \underline{-12} \\ 1 \end{array}$$

Para validar este problema, nuestro programa probador recibiría tanto el input como el output, y el proceso de validación se realizaría mediante:

$$\text{cociente} * \text{divisor} + \text{resto} = \text{dividendo}$$

$$972 * 6 + 1 = 5833 \rightarrow \text{TRUE}$$

Programa probador: complejidad - ejemplo 2

Veamos otro ejemplo, supongamos un sistema que calcula la inversa de una matriz, el procedimiento sería:

$$\begin{aligned}
 [A|I] &= \left[\begin{array}{ccc|ccc} 3 & -2 & 4 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{array} \right] \\
 &\sim \left[\begin{array}{ccc|ccc} 1 & 0 & 2 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 3 & -2 & 4 & 1 & 0 & 0 \end{array} \right] \\
 &\sim \left[\begin{array}{ccc|ccc} 1 & 0 & 2 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & -2 & -2 & 1 & -3 & 0 \end{array} \right] \\
 &\sim \left[\begin{array}{ccc|ccc} 1 & 0 & 2 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & -2 & 1 & -3 & 2 \end{array} \right] \\
 &\sim \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & -2 & 2 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1/2 & 3/2 & -1 \end{array} \right] = [I|A^{-1}]
 \end{aligned}$$

Programa probador: complejidad - ejemplo 2

Sin embargo, validar si la inversa la calculamos correctamente, no es más que multiplicar el input A por el output A^{-1} nos de como resultado la matriz identidad I (la cual es calculable).

En resumen, sería verificar que:

$$A \times A^{-1} = I$$

En este caso, realizar una multiplicación y luego una comparación de matrices, tiene complejidad bastante inferior al cálculo de la inversa de una matriz.

Programa probador - Salidas múltiples



En resumen, utilizar un programa probador, nos permite verificar que una determinada salida se corresponde con una entrada.

Esto tiene la ventaja de que si el problema original tiene salidas múltiples (o incluso infinitas salidas), no es ningún problema para un programa probador, ya que no realiza una comparación contra una salida precalculada como lo hace un lote de prueba.

Lote de prueba para el programa probador



Pero... ¿cómo garantizamos que el programa probador esté bien desarrollado? Es decir, podríamos tener errores en nuestro programa probador.

Para ello, debemos hacer un lote de prueba para nuestro programa probador, definiendo diferentes casos con combinaciones de inputs y outputs (que actuarían como entrada del lote, y la correspondiente salida para cada caso (true/false)).

Ejemplo de definición de caso de prueba

Nombre: caso_01_programa_probador_detecta_error

Descripción: Busca detectar un error en la validación por un resto inválido (que podría no estar manejado correctamente).

IN		OUT (esperado)
IN (original)	OUT (del sistema)	
5833 6	972 3	false

The background is a solid dark blue. Overlaid on this background is a faint, light blue illustration. It depicts a pen lying diagonally across the upper right portion of the frame. Below the pen, there are two rectangular notepads. Each notepad has a large, stylized checkmark drawn on it. The overall aesthetic is clean and professional, suggesting a theme of testing, approval, or quality control.

Lote de prueba

VS

Programa probador

Lote de prueba

~~VS~~ +

Programa probador

Lote de prueba + Programa probador



La primera conclusión que debemos sacar es que el lote de prueba y el programa probador son métodos de pruebas complementarios.

No debemos elegir entre uno y otro, sino que debemos utilizar el que más se ajuste a las características de nuestro problema, e incluso combinar ambos métodos para probar distintas características de nuestro sistema.

Por ejemplo, usar lote de prueba para casos con salidas únicas y programa probador para casos con salidas múltiples.

Lote de prueba + Programa probador



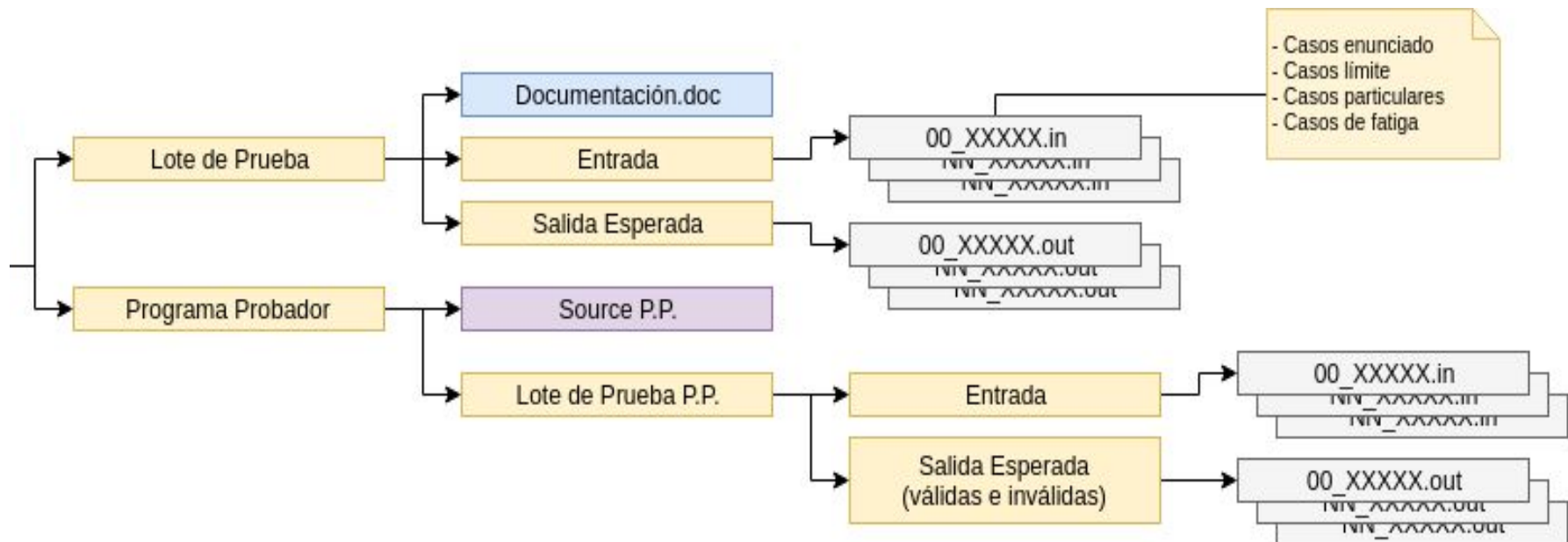
	Lote de Prueba	Programa Probador
Salidas únicas	✓	✓
Salidas múltiples	✗	✓
Identificación de errores	Si falla un caso, se sabe cuál podría ser el problema.	Si falla, no nos brinda información del problema.
Cálculo de salidas esperadas	Manual	Automático (programado)
Complejidad del problema original	Funciona para todo tipo de problemas	Debe tener una complejidad mucho menor

Organización de las pruebas

The background is a solid dark blue. Faint, light blue illustrations are scattered across the surface. On the right side, there is a detailed drawing of a pen. In the center and bottom right, there are several checkmarks, some of which are enclosed within rounded rectangular boxes. A dashed line is visible in the bottom left corner.

Organización de las pruebas

A continuación se muestra una propuesta de estructura de carpetas para organizar las pruebas:





Dudas

