PRUEBA. DEFINICIÓN
La prueba (testing) es el proceso de ejecutar un programa con la intención de encontrar errores.
Con el fin de desarrollar las pruebas de una manera óptima, los casos de prueba cobran especial importancia.

PRUEBAS
El programa se prueba ejecutando solo unos pocos casos de prueba dado que por lo general es física, económica o técnicamente imposible ejecutarlo para todos los valores de entrada posibles.

TÉCNICAS DE DISEÑO DE CASOS DE PRUEBAS
Clasificación clásica de técnicas de prueba:
Pruebas estructurales o de caja blanca: cuando se diseña la prueba a partir del conocimiento de la estructura interna del código. Estas pruebas se centran en la estructura interna del programa, analizando los caminos de ejecución. Problemas: No se prueba la especificación funcional y no se detectan ausencias
Pruebas funcionales o de caja negra: Su objetivo es validar que el código cumple la funcionalidad definida. Problemas: Infinitas posibilidades para las entradas.

TÉCNICAS DE DISEÑO DE CASOS DE PRUEBAS
Las **pruebas de caja blanca** permitirán recorrer todos los posibles caminos del código y ver qué sucede en cada caso posible. Se probará qué ocurre con las condiciones y los bucles que se ejecutan. Las pruebas se llevarán a cabo con datos que garanticen que han tenido lugar todas las combinaciones posibles. Para decidir qué valores deberán tomar estos datos es necesario saber cómo se ha desarrollado el código, buscando que no quede ningún rincón sin revisar.
Las técnicas de **pruebas de caja negra** pretenden encontrar errores en funciones incorrectas o ausentes, errores de interfaz, errores de rendimiento, inicialización y finalización. Se centra en las funciones y en sus entradas y salidas.
caja blanca, dentro de las pruebas unitarias, sirve para analizar el código en todas sus estructuras, en todos sus caminos del software. Pero existe otro tipo de pruebas que se basa en un enfoque más funcional, llamadas pruebas de **caja negra**.

PRUEBAS DE CAJA NEGRA
Habrá que elegir cuidadosamente los casos de prueba, de manera que sean tan pocos como sea posible para que la prueba se pueda ejecutar en un tiempo razonable y, al mismo tiempo, que cubran la variedad de entradas y salidas más amplia posible.
Para ello, se han diseñado diferentes técnicas:
Clases de equivalencia: se trata de determinar los diferentes tipos de entrada y salida, agruparlos y escoger casos de prueba para cada tipo o conjunto de datos de entrada y salida.
Análisis de los valores límite: estudian los valores iniciales y finales, ya que estadísticamente se ha demostrado que tienen más tendencia a detectar errores.
Estudio de errores típicos: la experiencia dice que hay una serie de errores que se suelen repetir en muchos programas; por ello, se trataría de diseñar casos de prueba que provocan las situaciones típicas de este tipo de errores.
Manejo de interfaz gráfica: para probar el funcionamiento de las interfaces gráficas, se deben diseñar casos de prueba que permitan descubrir errores en el manejo de ventanas, botones, iconos ...
Datos aleatorios: se trata de utilizar una herramienta que automatice las pruebas y que genere de forma aleatoria los casos de prueba. Esta técnica no optimiza la elección de los casos de prueba, pero si se hace durante bastante tiempo con muchos datos, podrá llegar a hacer una prueba bastante completa. Esta técnica se podría utilizar como complementaria a las anteriores o en casos en que no sea posible aplicar otra.
Clases de equivalencia
Se deben diseñar los casos de prueba de forma que prueben la mayor funcionalidad posible del programa, pero que no incluyan demasiados valores. ¿Por dónde empezar? ¿Qué valores se deben escoger?

1. **Identificar las condiciones** de las entradas y las salidas.
2. **Identificar**, a partir de las restricciones, las clases de equivalencia de las entradas y las salidas.

Por cada condición de entrada se identifican **clases de equivalencia válidas y no válidas**. Este proceso es heurístico. Sin embargo existen un conjunto de criterios que ayudan a su identificación:
Si una condición de entrada especifica un rango de valores para los datos de entrada, por ejemplo, si se admite del 10 al 50:

se creará una clase válida (10 ≤ X ≤ 50) y dos clases no válidas, una para los valores superiores (X> 50) y la otra para los inferiores (X <10).
Si una condición de entrada especifica un valor válido de entrado, por ejemplo, si la entrada comienza con mayúscula:
se crea una clase válida (con la primera letra mayúscula) y otra de no válida (con la primera letra minúscula).
Si se especifica un número de valores de entrada, por ejemplo, si se han de introducir tres números seguidos: se creará una clase válida (con tres valores) y dos de no válidas (una con menos de dos valores y el la otra con más de tres valores).
Si hay un conjunto de datos de entrada concretas válidas, se generará una clase para cada valor válido, por ejemplo, si la entrada debe ser rojo, naranja, verde: se generarán tres clases válidas y otra por un valor no válido (por ejemplo, azul).
Si no se han recogido ya con las clases anteriores, se debe seleccionar una clase para cada posible clase de resultado .

Resumen Clases de equivalencia

Identifica las condiciones de **entrada y salida.**
Identifica las **clases de equivalencia (Validas - Invalidas).**
Identifica los **casos de prueba (Escenario – Resultado Esperado).**

Análisis de valores límite

La experiencia muestra que los casos de prueba que exploran las **condiciones límite** producen mejor resultado que aquellos que no lo hacen.
Las condicione límite son aquellas que se hayan en los **márgenes (por encima y por debajo) de las clases de equivalencia**, tanto de entrada como de salida.
Por lo tanto, el análisis de valores límite complementa la técnica clases de equivalencia de manera que: al seleccionar el elemento representativo de la clase de equivalencia, en lugar de coger uno cualquiera, se escogen los valores al límite y, si se considera oportuno, un valor intermedio. Además, también se intenta que los valores en la entrada provoquen valores límite a los resultados.

RESUMEN CLASES DE EQUIVALENCIA Y VALORES LÍMITE

Todas las clases de equivalencia válidas e inválidas tienen unos valores lo suficientemente representativos de cada clase, qué, además explotan los límites operacionales.
Comprobamos si los resultados coinciden con los esperados.
Aún así no podemos garantizar que el programa esté 100% libre de errores.

Errores típicos

Para diseñar casos de prueba, también se puede aprovechar la **experiencia previa**. Hay una serie de errores que se repiten mucho en los programas, y podría ser una buena estrategia utilizar casos de prueba que se centren en buscar estos errores. De este modo, se mejorará la elección de los representantes de las clases de equivalencia:
El valor **cero** suele provocar errores, por ejemplo, una división por cero aborta el programa. Si se tiene la posibilidad de introducir ceros a la entrada, se debe escoger en los casos de prueba.
Cuando se debe introducir una **lista de valores**, habrá que centrarse en la posibilidad de no i**ntroducir ningún valor**, o introducir **uno**.
Hay que pensar que el **usuario puede introducir entradas que no son normales**, por eso es recomendable ponerse en el peor caso.
Los **desbordamientos de memoria** son habituales, por eso se debe intentar introducir **valores tan grandes como sea posible**.

Manejo de interfaz gráfica

No sólo se debe hablar de entradas de textos, también hay que tener en cuenta los entornos gráficos donde se llevan a cabo las entradas de valores o donde se visualizan los resultados. Actualmente, la mayoría de programas suelen interactuar con el usuario haciendo uso de sistemas gráficos que cada vez son más complejos, con lo cual se pueden generar errores.
Las **pruebas** de interfaz gráfica de usuario deben incluir:
Pruebas sobre ventanas: iconos de cerrar, minimizar ...
Pruebas sobre menús y uso de ratón.
Pruebas de entrada de datos: cuadro de textos, listas
desplegables ...
Pruebas de documentación y ayuda del programa.
Otros.

RESUMEN

El proceso de prueba consiste en ejecutar el programa con el fin de localizar errores.

La prueba es una actividad incompleta.

Propósito de las técnicas de diseño de casos de prueba es **reducir el número de casos de prueba sin mermar la efectividad de la prueba**.

Existen dos enfoques a la hora de abordar el diseño de los casos de prueba:

Caja Negra: evalúa la funcionalidad del sistema (lo que hace).

Caja Blanca: evalúa la lógica interna (como lo hace).