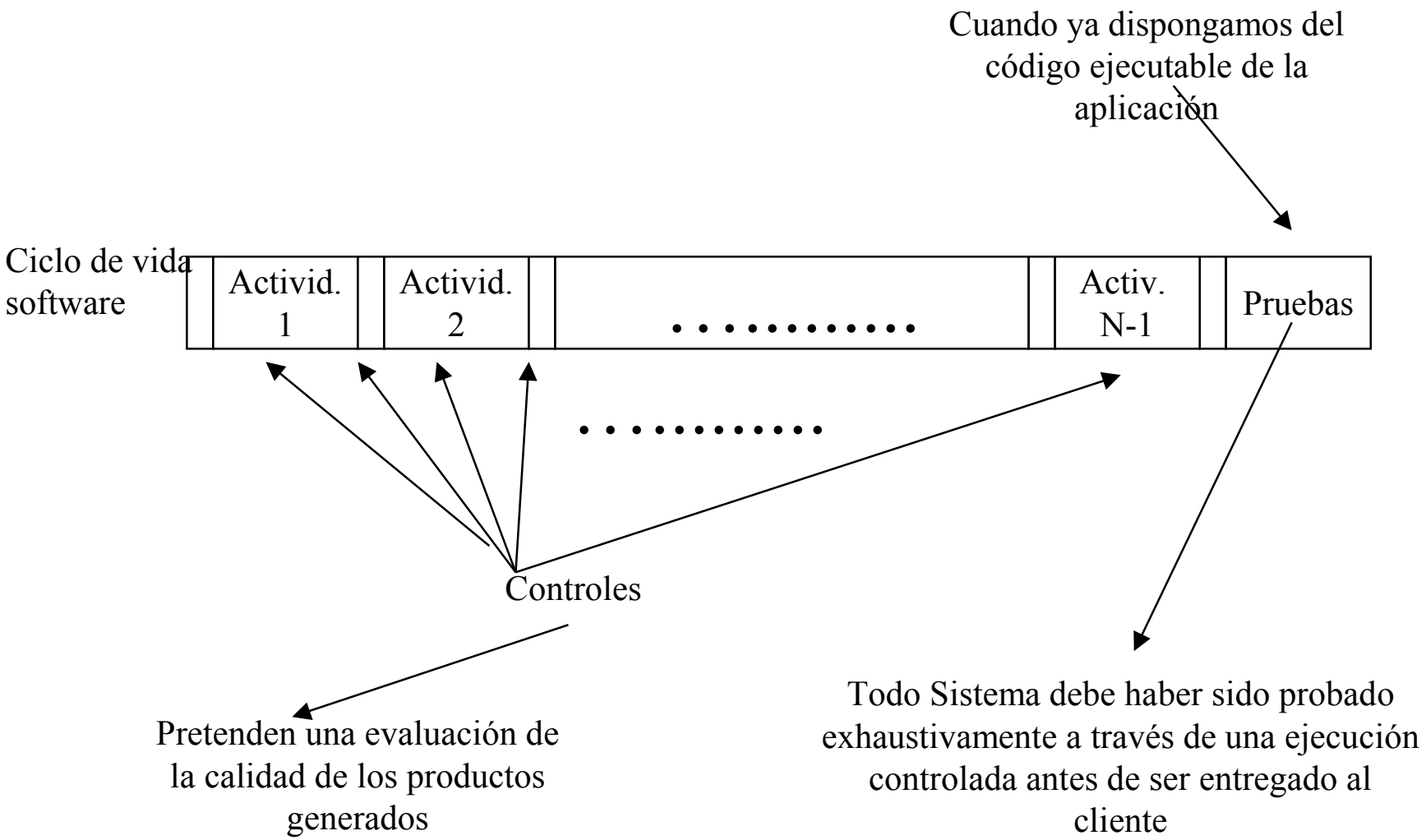
Tema 9. Pruebas del Software

1. Definiciones asociadas
2. El proceso de prueba
3. Técnicas de diseño de casos de prueba
4. Pruebas estructurales
5. Pruebas funcionales
6. Pruebas aleatorias
7. Enfoque práctica de diseño de casos
8. Documentación del diseño de pruebas
9. Ejecución de pruebas
10. Estrategia de aplicación de pruebas en el ciclo de vida

PRUEBAS DEL SOFTWARE

12.005



✉ **Verificación:** El proceso de evaluación de un sistema (o de uno de sus componentes para determinar si los productos de una fase dada satisfacen las condiciones impuestas al comienzo de dicha fase

¿Estamos construyendo correctamente el producto?

¿Estamos construyendo el producto correcto?

✉ **Validación:** El proceso de evaluación de un sistema o de uno de sus componentes durante o al final del proceso de desarrollo para determinar si satisface los requisitos marcados por el usuario

DEFINICIONES

- **Pruebas** *(test):* «una actividad en la cual un sistema o uno de sus componentes se ejecuta en circunstancias previamente especificadas, los resultados se observan y registran y se realiza una evaluación de algún aspecto»

Proceso de ejecutar un programa con el fin de encontrar errores

- **Caso de prueba** *(test case):* «un conjunto de entradas, condiciones de ejecución y resultados esperados desarrollados para un objetivo particular»

- **Defecto** *(defect, fault, «bug»):* «un defecto en el software como, por ejemplo, un proceso, una definición de datos o un paso de procesamiento incorrectos en un programa»

Instrucción incorrecta

DEFINICIONES

- **Fallo** *(failure):* «La incapacidad de un sistema o de alguno de sus componentes para realizar las funciones requeridas dentro de los requisitos de rendimiento especificados»
- **Error** *(error):* tiene varias acepciones:

Fallo ←

- La diferencia entre un valor calculado, observado o medio y el valo verdadero, especificado o teóricamente correcto.

Defecto ←

- Un defecto

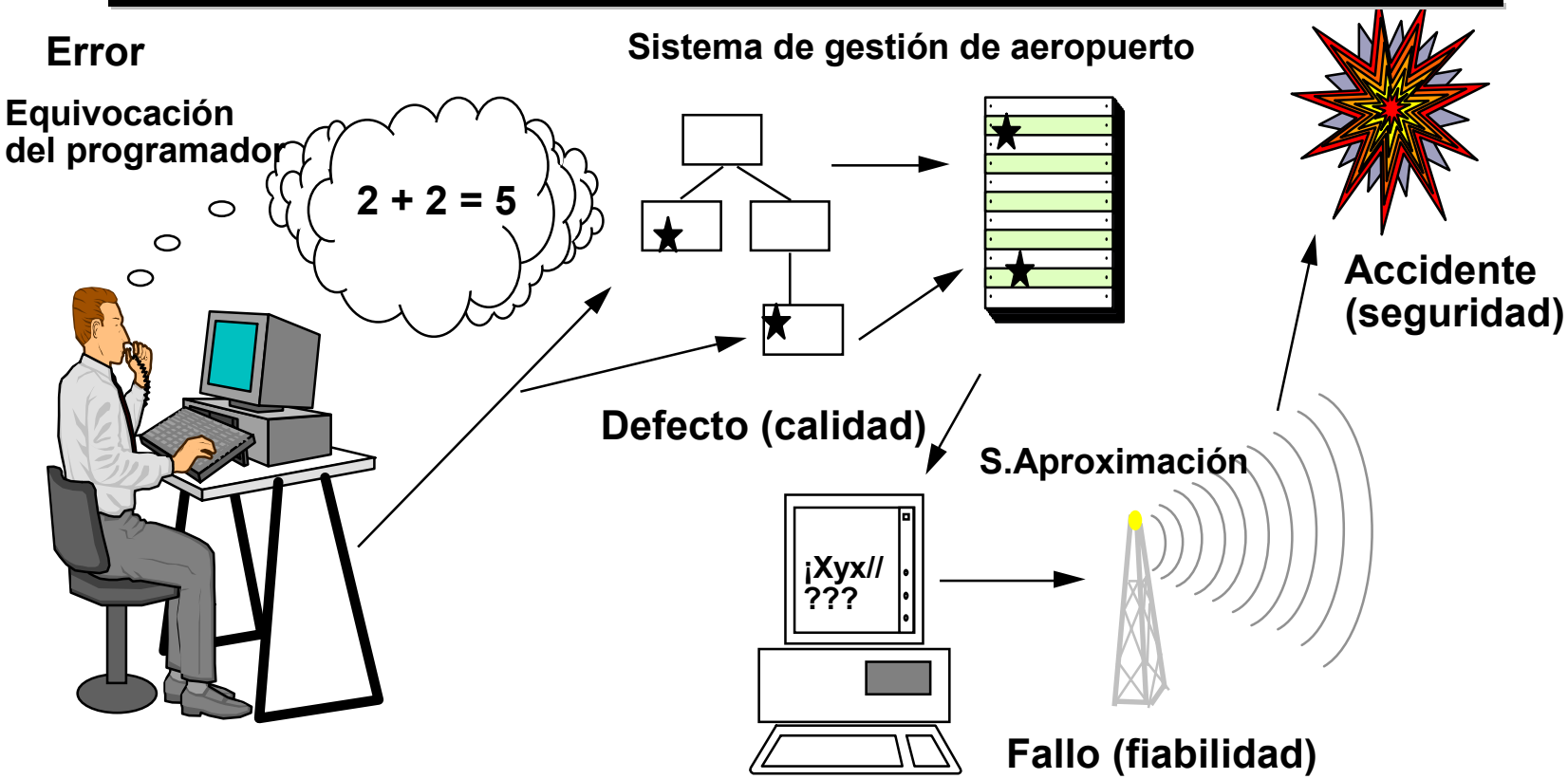
Fallo ←

- Un resultado incorrecto

Error ←

- Una acción humana que conduce a un resultado incorrecto .

RELACION ENTRE ERROR, DEFECTO Y FALLO



Error (del programador) —Se Plasma→ Defecto (en el software) —Da lugar a→ Fallo (el sistema no se comporta como debería) —Que provoca→ Efectos negativos (dependiendo de la criticidad del sistema)

IDEAS PARADÓGICAS DE LAS PRUEBAS

- La prueba exhaustiva del software es impracticable (no se pueden probar todas las posibilidades de su funcionamiento ni siquiera en programas sencillos
- El objetivo de las pruebas es la detección de defectos en el software (descubrir un error es el éxito de una prueba)
  - Mito → un defecto implica que somos malos profesionales y que debemos sentirnos culpables → todo el mundo comete errores
- El descubrimiento de un defecto significa un éxito para la mejora de la calidad

RECOMENDACIONES PARA UNAS PRUEBAS EXITOSAS

- Cada caso de prueba debe definir el resultado de salida esperado que se comparará con el realmente obtenido.

- El programador debe evitar probar sus propios programas, ya que desea (consciente o inconscientemente) demostrar que funcionan sin problemas.
  - → Además, es normal que las situaciones que olvidó considerar al crear el programa queden de nuevo olvidados al crear los casos de prueba

- Se debe inspeccionar a conciencia el resultado de cada prueba, así, poder descubrir posibles síntomas de defectos.

- Al generar casos de prueba, se deben incluir tanto datos de entrada válidos y esperados como no válidos e inesperados.

RECOMENDACIONES PARA UNAS PRUEBAS EXITOSAS

- Las pruebas deben centrarse en dos objetivos (es habitual olvidar el segundo):
  - Probar si el software no hace lo que debe hacer
  - Probar si el software hace lo que debe hacer, es decir, si provoca efectos secundarios adversos
- Se deben evitar los casos desechables, es decir, los no documentados ni diseñados con cuidado.
  - → Ya que suele ser necesario probar muchas veces el software y por tanto hay que tener claro qué funciona y qué no
- No deben hacerse planes de prueba suponiendo que, prácticamente, no hay defectos en los programas y, por lo tanto, dedicando pocos recursos a las pruebas → **siempre hay defectos**

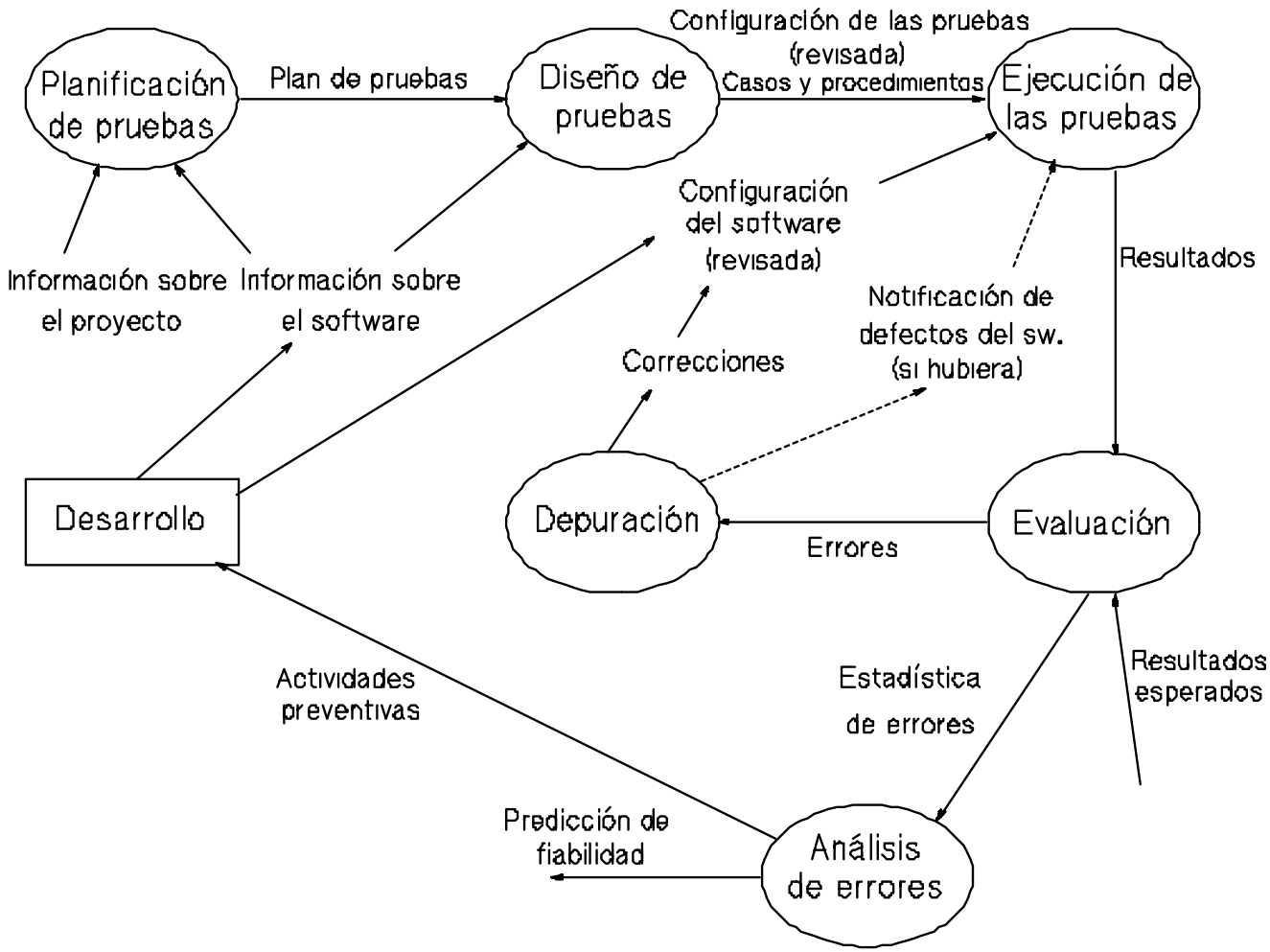
RECOMENDACIONES PARA UNAS PRUEBAS EXITOSAS

- La experiencia parece indicar que donde hay un defecto hay otros, es decir, la probabilidad de descubrir nuevos defectos en una parte del software es proporcional al número de defectos ya descubierto.

- Las pruebas son una tarea tanto o más creativa que el desarrollo de software. Siempre se han considerado las pruebas como una tarea destructiva y rutinaria.

Es interesante planificar y diseñar las pruebas para poder detectar el máximo número y variedad de defectos con el mínimo consumo de tiempo y esfuerzo

CICLO COMPLETO DE LAS PRUEBAS



PROCESO DE PRUEBA

ACTIVIDADES

↓ La depuración (localización y corrección de defectos)

↓ El análisis de la estadística de errores

→ Sirve para realizar predicciones de la fiabilidad del software y para detectar las causas más habituales de error y por tanto mejorar los procesos de desarrollo

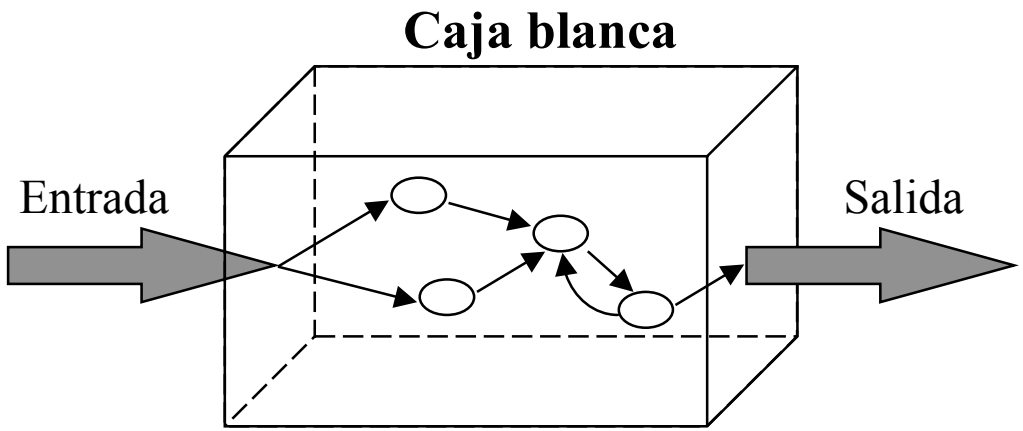
ENFOQUES DE DISEÑO DE PRUEBAS

Existen **tres** enfoques principales para el diseño de casos:

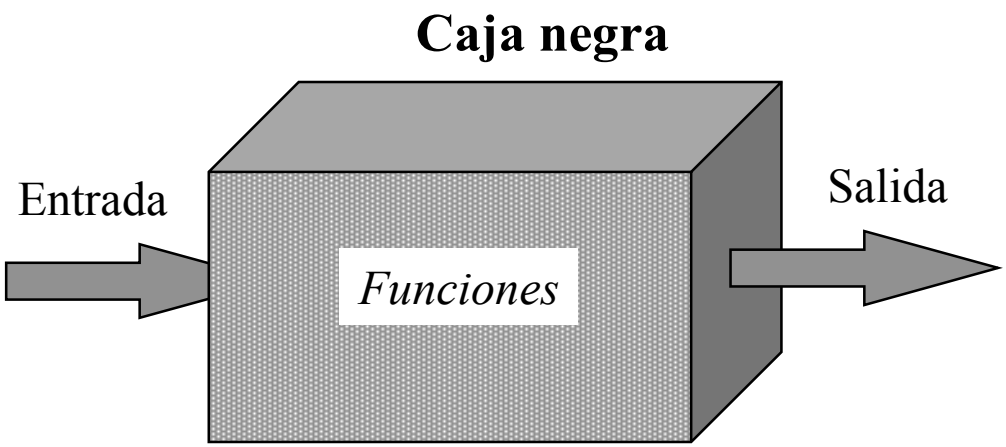
1.- El enfoque **estructural** o de **caja blanca.** Se centra en la estructura interna del programa (analiza los caminos de ejecución).
2.- El enfoque **funcional** o de **caja negra.** Se centra en las funciones, entradas y salidas.
3.- El enfoque **aleatorio** consiste en utilizar modelos (en muchas ocasiones estadísticos) que representen las posibles entradas al programa para crear a partir de ellos los casos de prueba

LOS ENFOQUES DE DISEÑO DE PRUEBAS DE CAJA BLANCA Y DE CAJA NEGRA

Estructural



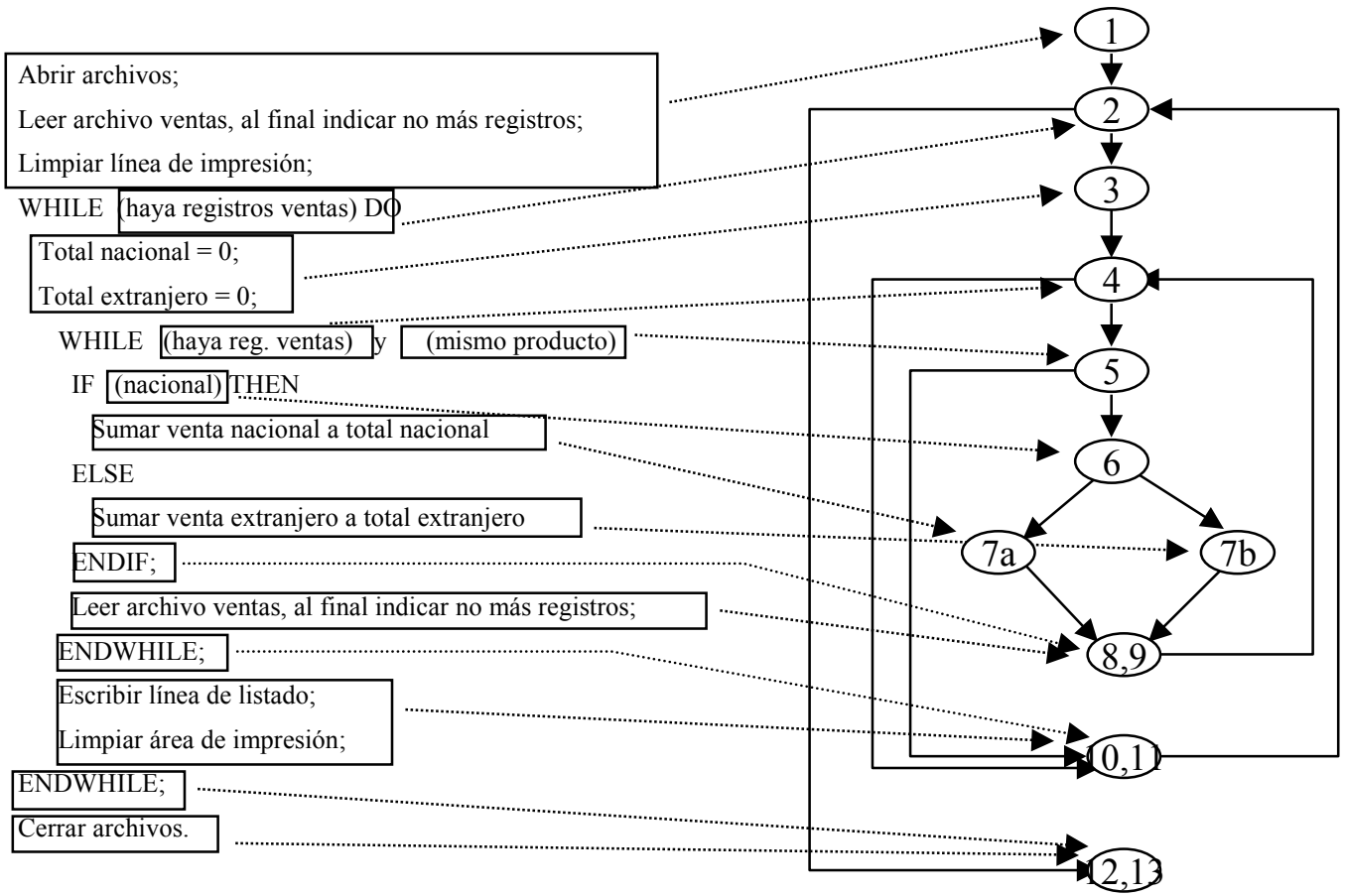
Funcional



PRUEBAS ESTRUCTURALES

GRAFO DE FLUJO DE UN PROGRAMA (PSEUDOCODIGO)

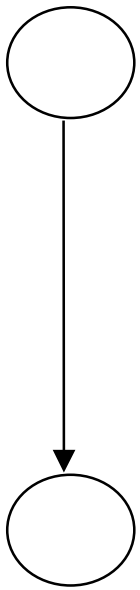
El diseño de casos de prueba tiene que estar basado en la elección de caminos importantes que ofrezcan una seguridad aceptable



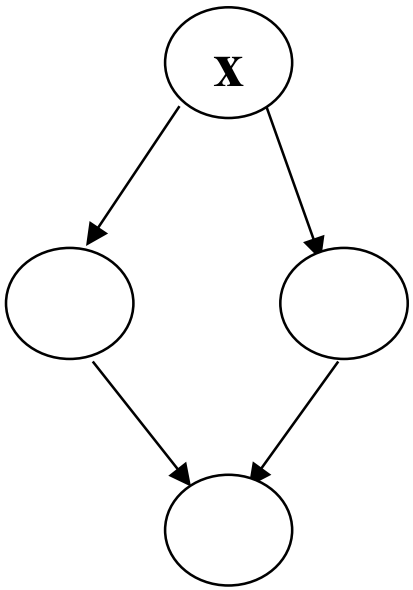
de que se descubren defectos (un programa de 50 líneas con 25 sentencias if en serie da lugar a 33,5 millones de secuencias posibles), para lo que se usan los criterios de cobertura lógica.

PRUEBAS ESTRUCTURALES

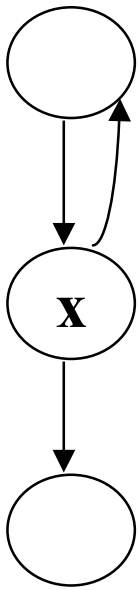
GRAFO DE FLUJO DE LAS ESTRUCTURAS BASICAS DE PROGRAMA



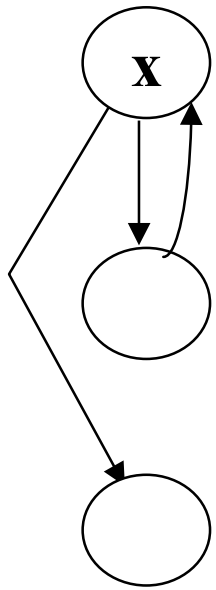
Secuencia



Si x entonces...
(If x then...else...)



Hacer... hasta x
(Do...until x)
Repetir



Mientras x hacer...
(While x do...)

Consejos:

-Separar todas las condiciones

-Agrupar sentencias 'simples' en bloques

-Numerar todos los bloques de sentencias y también las condiciones

Menos riguroso

(más barato)

PRUEBAS ESTRUCTURALES

CRITERIOS DE COBERTURA LÓGICA

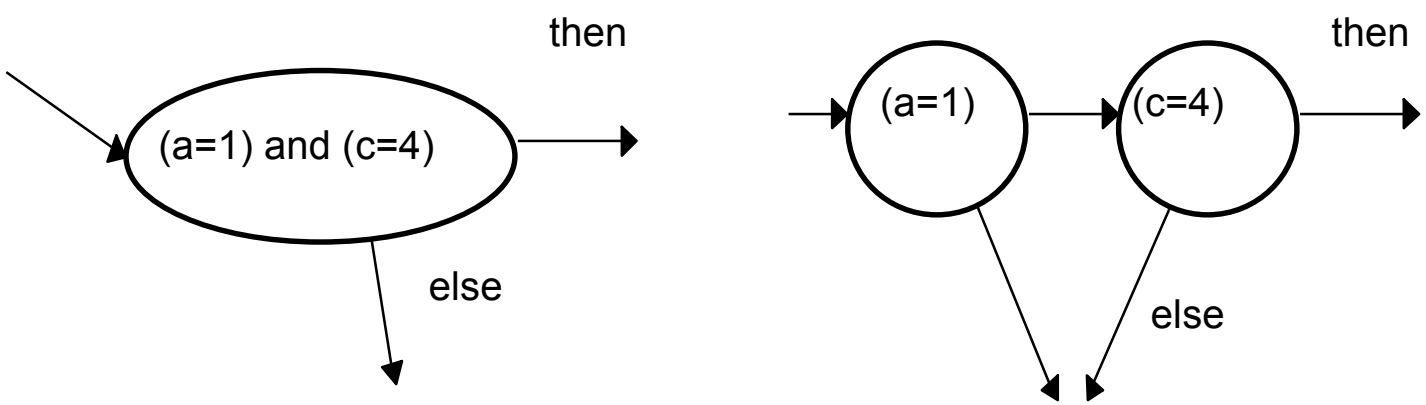
- ☑ **Cobertura de sentencias**. Se trata de generar los casos de prueba necesarios para que cada sentencia o instrucción del programa se ejecute al menos una vez.
- ☑ **Cobertura de decisiones**. Consiste en escribir casos suficientes para que cada decisión tenga, por lo menos una vez, un resultado verdadero y, al menos una vez, uno falso. (Incluye a la cobertura de sentencias)
- ☑ **Cobertura de condiciones**. Se trata de diseñar tantos casos como sea necesario para que cada condición de cada decisión adopte el valor verdadero al menos una vez y el falso al menos una vez. (No incluye cobertura de condiciones)
- ☑ **Criterio de decisión/condición**. Consiste en exigir el criterio de cobertura de condiciones obligando a que se cumpla también el criterio de decisiones.
- ☑ **Criterio de condición múltiple**. En el caso de que se considere que la evaluación de las condiciones de cada decisión no se realiza de forma simultánea, se puede considerar que cada decisión multicondicional se descompone en varias condiciones unicondicionales. Ejemplo en la siguiente diapositiva.
- ☑ **Criterio de cobertura de caminos**. Se recorren todos los caminos (impracticable)

Más riguroso

(más caros)

PRUEBAS ESTRUCTURALES

EJEMPLO DE DESCOMPOSICION DE UNA DECISION MULTICONDICIONAL



PRUEBAS ESTRUCTURALES

LA COMPLEJIDAD DE McCABE V(G) (COMPLEJIDAD CICLOMÁTICA)

- La métrica de McCabe ha sido muy popular en el diseño de pruebas
- Es un indicador del número de caminos independientes que existen en un grafo

La complejidad de McCabe se puede calcular de cualquiera de estas 3 formas

1. V (G) = *a* - *n* + 2, siendo *a* el número de arcos o aristas del grafo y *n* el número de nodos.
2. V (G) = *r*, siendo *r* el número de regiones cerradas del grafo.
3. V(G) = *c* + 1, siendo *c* el número de nodos de condición.

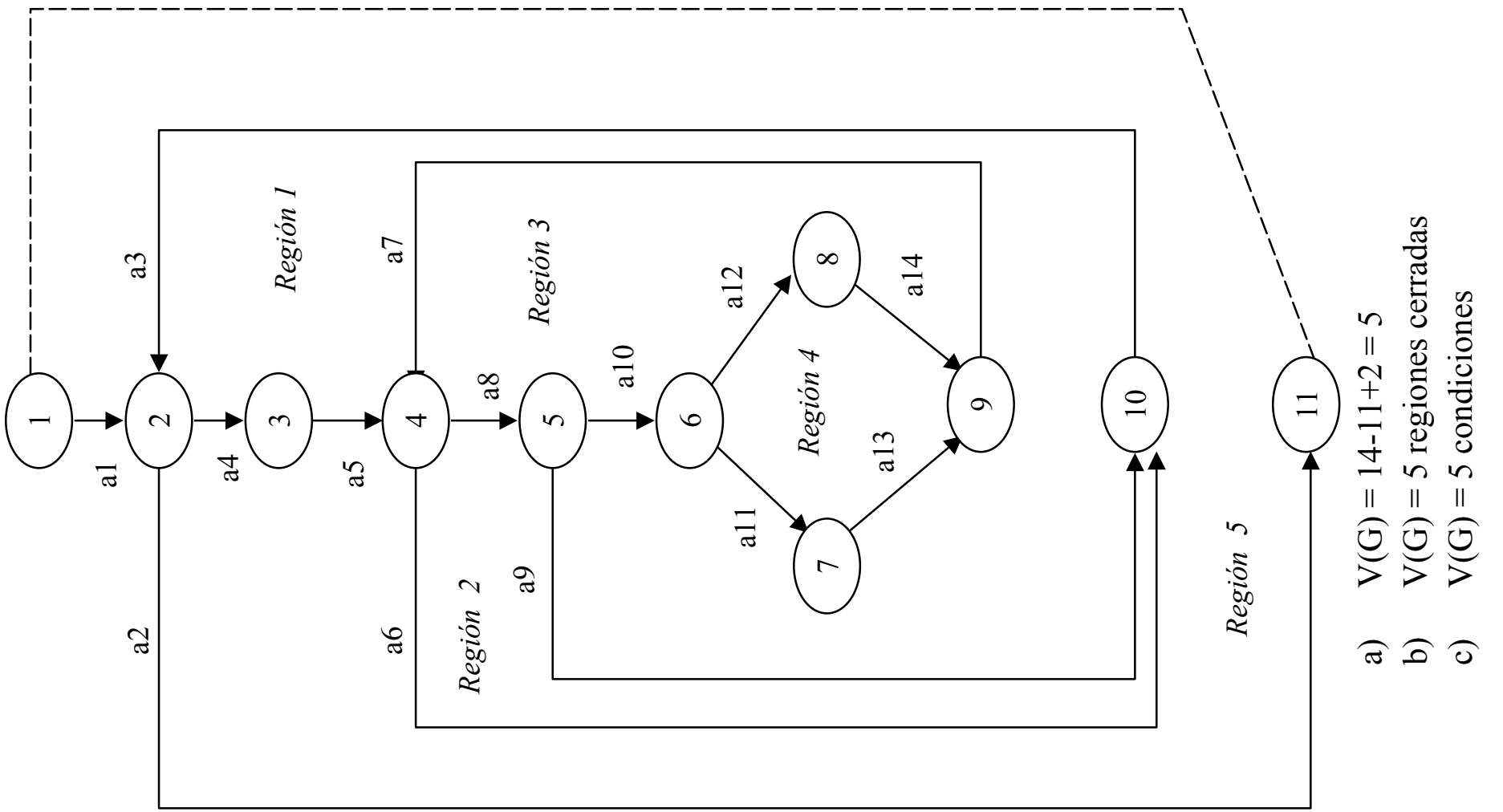
PRUEBAS ESTRUCTURALES

LA COMPLEJIDAD DE McCABE V(G)

- El criterio de prueba de McCabe es: Elegir tantos casos de prueba como caminos independientes (calculados como V(G))
- La experiencia en este campo asegura que:
  - V(G) marca el límite mínimo de casos de prueba para un programa
  - Cuando V(G) > 10 la probabilidad de defectos en el módulo o programa crece mucho → quizás sea interesante dividir el módulo

PRUEBAS ESTRUCTURALES

CALCULO DE LA COMPLEJIDAD CICLOMATICA SOBRE UN GRAFO



PRUEBAS FUNCIONALES

Se centran en las funciones, entradas y salidas → Es impracticable probar el software para todas las posibilidades. De nuevo hay que tener criterios para elegir buenos casos de prueba

Un caso de prueba funcional es bien elegido si se cumple que:

- Reduce el número de otros casos necesarios para que la prueba sea razonable. Esto implica que el caso ejecute el máximo número de posibilidades de entrada diferentes para así reducir el total de casos.
- Cubre un conjunto extenso de otros casos posibles, es decir, no indica algo acerca de la ausencia o la presencia de defectos en el conjunto específico de entradas que prueba, así como de otros conjuntos similares.

PRUEBAS FUNCIONALES

TÉCNICA: PARTICIPACIONES O CLASES DE EQUIVALENCIA

Cualidades que definen un buen caso de prueba

- Cada caso debe cubrir el máximo número de entradas
- Debe tratarse el dominio de valores de entrada dividido en un número finito de clases de equivalencia que cumplan la siguiente propiedad: la prueba de un valor representativo de una clase permite suponer «razonablemente» que el resultado obtenido (existan defectos o no) será el mismo que el obtenido probando cualquier otro valor de la clase

Lo que hay que hacer es:

- Identificar clases de equivalencia
- Crear casos de prueba correspondiente

PRUEBAS FUNCIONALES

PARTICIPACIONES O CLASES DE EQUIVALENCIA

PASOS PARA IDENTIFICAR CLASES DE EQUIVALENCIA

1. Identificación de las condiciones de las entradas del programa, es decir, restricciones de formato o contenido de los datos de entrada
2. A partir de ellas, se identifican clases de equivalencia que pueden ser:

   a) De datos válidos

   b) De datos no válidos o erróneos
3. Existen algunas reglas que ayudan a identificar clase:

   a) Si se especifica un rango de valores para los datos de entrada, secreará una clase válida y dos clases no válidas

   b) Si se especifica un número finito y consecutivo de valores, se creará una clase válida y dos no válidas

PRUEBAS FUNCIONALES

PARTICIPACIONES O CLASES DE EQUIVALENCIA

PASOS PARA IDENTIFICAR CLASES DE EQUIVALENCIA

c) Si se especifica una situación del tipo «debe ser» o booleana (por ejemplo, «el primer carácter debe ser una letra»), se identifican una clase válida («es una letra») y una no válida («no es una letra»)

d) Si se especifica un conjunto de valores admitidos y se sabe que el programa trata de forma diferente cada uno de ellos, se identifica una clase válida por cada valor y una no válida

e) En cualquier caso, si se sospecha que ciertos elementos de una clase no se tratan igual que el resto de la misma, deben dividirse en clases menores

PRUEBAS FUNCIONALES

PARTICIPACIONES O CLASES DE EQUIVALENCIA

PASOS PARA IDENTIFICAR CASOS DE PRUEBA

El último paso del método es el uso de las clases de equivalencia para identificar los casos de prueba correspondientes. Este proceso consta de las siguientes fases:

- Asignación de un número único a cada clase de equivalencia
- Hasta que todas las clases de equivalencia válidas hayan sido cubiertas por (incorporadas a) casos de prueba, se tratará de escribir un caso que cubra tantas clases válidas no incorporadas como sea posible

- Hasta que todas las clases de equivalencia no válidas hayan sido cubiertas por casos de prueba, escribir un caso para una única clase no válida sin cubrir

PRUEBAS FUNCIONALES

TABLA DE CLASES DE EQUIVALENCIA DEL EJEMPLO

Ejemplo: Aplicación bancaria en la que el operador debe proporcionar un código, un nombre y una operación

Condición de entrada	Clases válidas	Clases inválidas
Código área Nº de 3 dígitos que no empieza con 0 ni 1)	(1) 200 ≤código ≤999	(2) código <200 (3) código >999 (4) no es número
Nombre para identificar la operación	(5) seis caracteres	(6) menos de 6 caracteres (7) más de 6 caracteres
Orden Una de las siguientes →	(8) «cheque» (9) «depósito» (10) «pago factura» (11) «retirada de fondos»	(12) ninguna orden válida

Habría que diseñar casos de prueba que cubran todas las clases de equivalencia, tanto válidas como inválidas, y para las inválidas en casos de prueba distintos

PRUEBAS FUNCIONALES
TÉCNICA: ANALISIS DE VALORES LIMITE (AVL)

- La experiencia indica que los casos de prueba que exploran las condiciones límite de un programa producen un mejor resultado para detectar defectos
- El AVL es una técnica de diseño de casos de prueba que complementa a la de particiones de equivalencia. Las diferencias son las siguientes:

  - Más que elegir «cualquier» elemento como representativo de una clase de equivalencia, se requiere la selección de uno o más elementos tal que los márgenes se sometan a prueba
  - Más que concentrarse únicamente en el dominio de entrada (condiciones de entrada), los casos de prueba se generan considerando también el espacio de salida

PRUEBAS FUNCIONALES

ANALISIS DE VALORES LIMITE (AVL)

LAS REGLAS PARA IDENTIFICAR CLASES SON:

1. Si una condición de entrada especifica un rango de valores, se deben generar casos para los extremos del rango y casos no válidos para situaciones justo más allá de los extremos
2. Si la condición de entrada especifica un número finito y consecutivo de valores, hay que escribir casos para los números máximo, mínimo, uno más del máximo y uno menos del mínimo de valores
3. Usar la regla 1 para la condición de salida
4. Usar la regla 2 para cada condición de salida

   - *Los valores límite de entrada no generan necesariamente los valores límite de salida (recuérdese la función seno, por ejemplo)*
   - *No siempre se pueden generar resultados fuera del rango de salida (pero es interesante considerarlo)*

5. Si la entrada o la salida de un programa es un conjunto ordenado, los casos se deben concentrar en el primero y en el último elemento

PRUEBAS FUNCIONALES
TÉCNICA: CONJETURA DE ERRORES

Se enumera una lista de posibles equivocaciones típicas que pueden cometer los desarrolladores y de situaciones propensas a ciertos errores

- ☑ El valor cero es una situación propensa a error tanto en la salida como en la entrada
- ☑ En situaciones en las que se introduce un número variable de valores, conviene centrarse en el caso de no introducir ningún valor y en el de un solo valor. También puede ser interesante un lista que tiene todos los valores iguales
- ☑ Es recomendable imaginar que el programador pudiera haber interpretado algo mal en la especificación
- ☑ También interesa imaginar lo que el usuario puede introducir como entrada a un programa

•••

PRUEBAS ALEATORIAS

En las pruebas aleatorias simulamos la entrada habitual del programa creando datos de entrada en la secuencia y con la frecuencia con las que podrían aparecer en la Práctica (de manera repetitiva)

Para ello habitualmente se utilizan generadores automáticos de casos de prueba

ENFOQUE PRACTICO RECOMENDADO PARA EL DISEÑO DE CASOS

Se propone un uso más apropiado de cada técnica (caja blanca y negra) para obtener un conjunto de casos útiles:

- *Si la especificación contiene combinaciones de condiciones de entrada, comenzar formando sus grafos de causa-efecto (ayudan a la comprensión de dichas combinaciones)*

- *En todos los casos, usar el análisis de valores límites para añadir casos de prueba: elegir límites para dar valores a las causas en los casos generados asumiendo que cada causa es una clase de equivalencia*

- *Identificar las clases válidas y no válidas de equivalencia para la entrada y la salida, y añadir los casos no incluidos anteriormente*

ENFOQUE PRACTICO RECOMENDADO PARA EL DISEÑO DE CASOS

- *Utilizar la técnica de conjetura de errores para añadir nuevos casos, referidos a valores especiales*

- *Ejecutar los casos generados hasta el momento y analizar la cobertura obtenida*

- *Examinar la lógica del programa para añadir los casos precisos (de caja blanca) para cumplir el criterio de cobertura elegido si los resultados de la ejecución del punto anterior indican que no se ha satisfecho el criterio de cobertura elegido*

Una cuestión importante es ¿por qué son necesarias las pruebas de caja blanca si comprobamos que las funciones se realizan correctamente?

- *Los errores lógicos y las suposiciones incorrectas son inversamente proporcionales a la probabilidad de que se ejecute un camino del programa (a menor probabilidad de ejecutarse un camino, mayor número de errores)*
- *Se suele creer que un determinado camino lógico tiene pocas posibilidades de ejecutarse cuando, de hecho, se puede ejecutar regularmente*
- *Los errores tipográficos son aleatorios; pueden aparecer en cualquier parte del programa (sea muy usada o no)*
- *La probabilidad y la importancia de un trozo de código suele ser calculada de forma muy subjetiva*

No obstante las pruebas de caja blanca sólas tampoco garantizan el éxito:

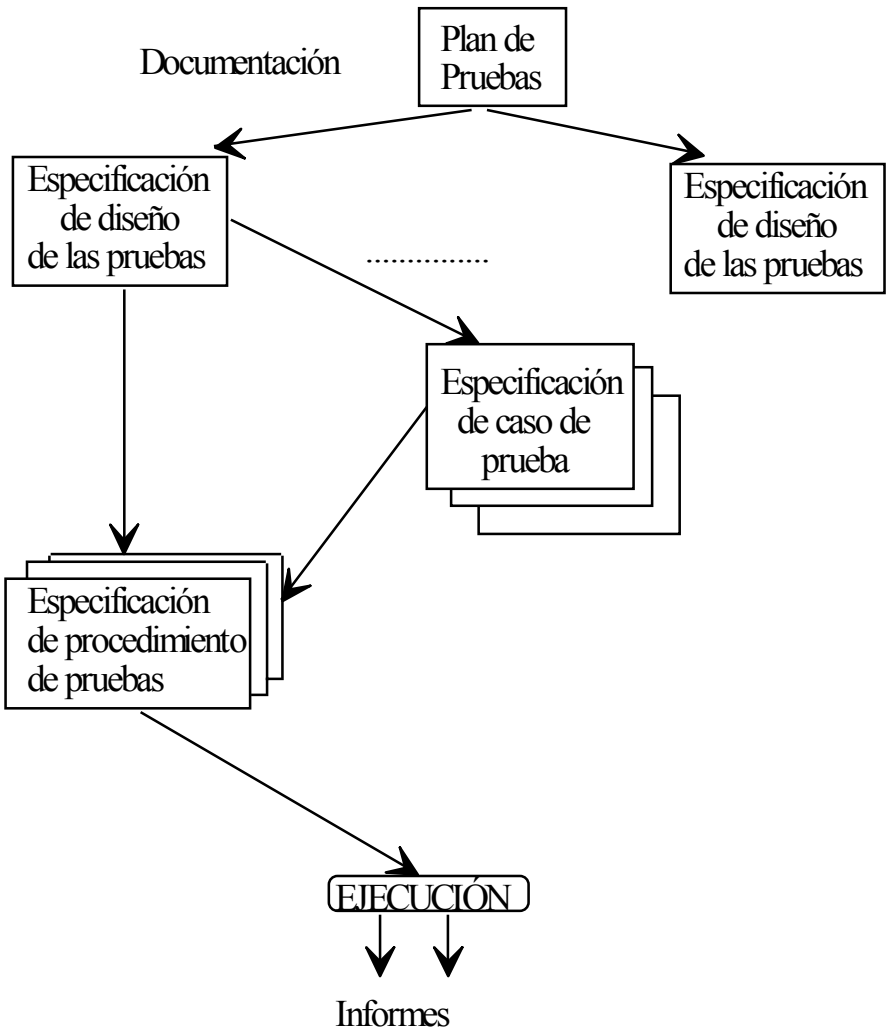
if (x+y+z)/3=x then print (x,y y z son iguales) else print (x,y y z no son iguales)

Una prueba de caja blanca como esta

x=5 y=5 z=5 x=2 y=3 z=7^12

No detecta el error lógico, que si sería detectado con otro tipo de pruebas funcionales.

DOCUMENTOS RELACIONADOS CON EL DISEÑO DE LAS PRUEBAS SEGUN EL ESTANDAR IEEE std. 829



PLAN DE PRUEBAS

Objetivo del documento

Señalar el enfoque, los recursos y el esquema de actividades de prueba, así como los elementos a probar, las características, las actividades de prueba, el personal responsable y los riesgos asociados

PLAN DE PRUEBAS

Estructura fijada en el estándar

1. *Identificador único del documento*
2. *Introducción y resumen de elementos y características a probar*
3. *Elementos software a probar*
4. *Características a probar*
5. *Características que no se probarán*
6. *Enfoque general de la prueba*
7. *Criterios de paso/fallo para cada elemento*
8. *Criterios de suspensión y requisitos de reanudación*
9. *Documentos a entregar*
10. *Actividades de preparación y ejecución de pruebas*
11. *Necesidades de entorno*
12. *Responsabilidades en la organización y realización de las pruebas*
13. *Necesidades de personal y formación*
14. *Esquema de tiempos*
15. *Riesgos asumidos por el plan y planes de contingencias*
16. *Aprobaciones y firmas con nombre y puesto desempeñado*

ESPECIFICACION DEL DISEÑO DE PRUEBAS

Objetivo del documento

Especificar los refinamientos necesarios sobre el enfoque general reflejado en el plan e identificar las características que se deben probar con este diseño de pruebas

ESPECIFICACION DEL DISEÑO DE PRUEBAS

Estructura fijada en el estándar

- *Identificador único para la especificación. Proporcionar también una referencia del plan asociado (si existe)*
- *Características a probar de los elementos software (y combinaciones de características)*
- *Detalles sobre el plan de pruebas del que surge este diseño, incluyendo las técnicas de prueba específica y los métodos de análisis de resultados*
- *Identificación de cada prueba:*
  - *identificador*
  - *casos que se van a utilizar*
  - *procedimientos que se van a seguir*
- *Criterios de paso/fallo de la prueba (criterios para determinar si una característica o combinación de características ha pasado con éxito la prueba o no)*

ESPECIFICACION DE CASO DE PRUEBA

Objetivo del documento

Definir uno de los casos de prueba identificando por una especificación del diseño de las pruebas

ESPECIFICACION DE CASO DE PRUEBA

Estructura fijada en el estándar

- ✉ *Identificador único de la especificación*
- ✉ *Elementos software (por ejemplo, módulos) que se van a probar: definir dichos elementos y las características que ejercitará este caso*
- ✉ *Especificaciones de cada entrada requerida para ejecutar el caso (incluyendo las relaciones entre las diversas entradas; por ejemplo, la sincronización de las mismas)*
- ✉ *Especificaciones de todas las salidas y las características requeridas (por ejemplo, el tiempo respuesta) para los elementos que se van a probar*
- ✉ *Necesidades de entorno (hardware, software y otras como, por ejemplo, el personal)*
- ✉ *Requisitos especiales de procedimiento (o restricciones especiales en los procedimientos para ejecutar este caso).*
- ✉ *Dependencias entre casos (por ejemplo, listar los identificadores de los casos que se van a ejecutar antes de este caso de prueba)*

ESPECIFICACION DE PROCEDIMIENTO DE PRUEBA

Objetivo del documento

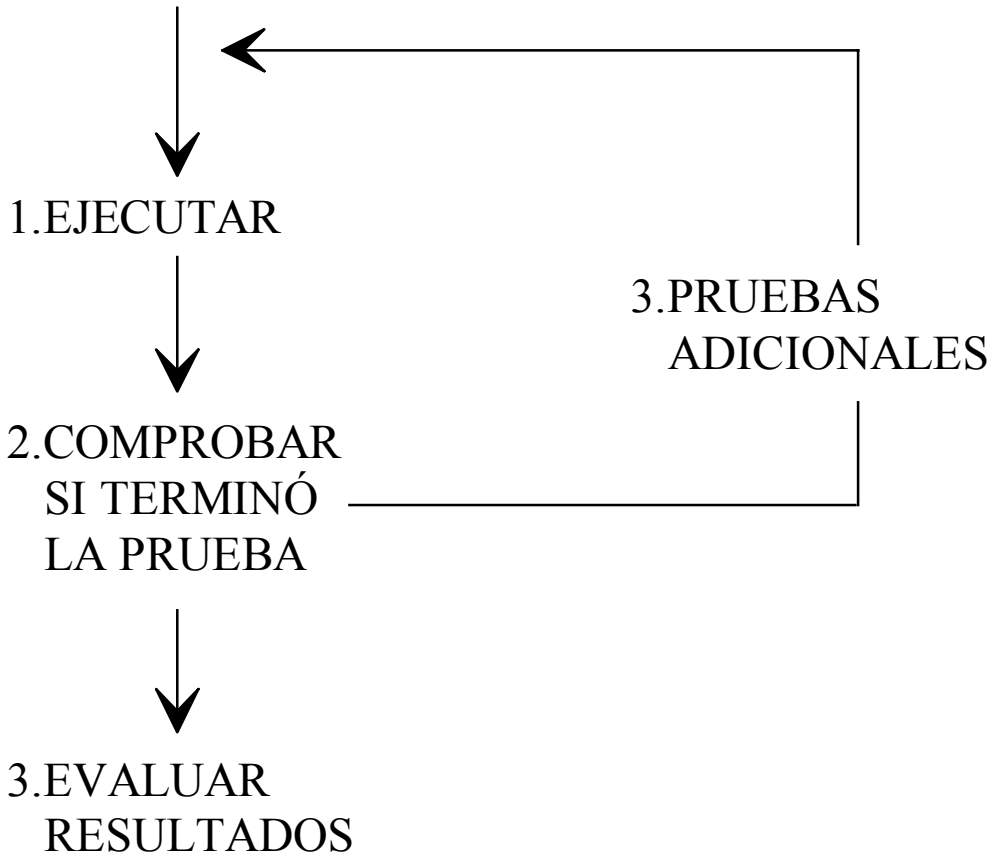
Especificar los pasos para la ejecución de un conjunto de casos de prueba o, más generalmente, los pasos utilizados para analizar un elemento software con el propósito de evaluar un conjunto de características del mismo

ESPECIFICACION DE PROCEDIMIENTO DE PRUEBA

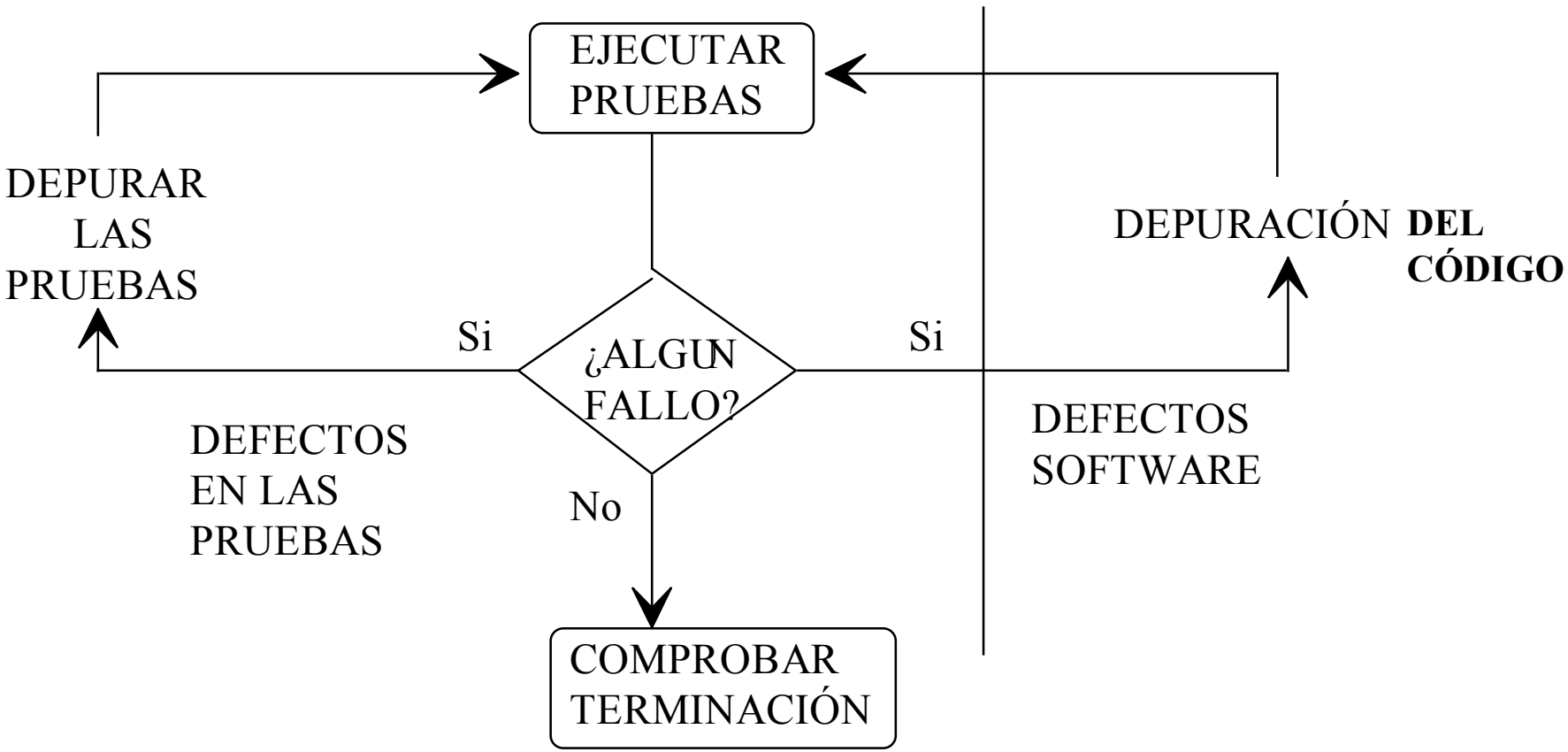
Estructura fijada en el estándar

- *Identificador único de la especificación y referencia a la correspondiente especificación de diseño de prueba*
- *Objetivo del procedimiento y lista de casos que se ejecutan con él*
- *Requisitos especiales para la ejecución (por ejemplo, entorno especial o personal especial)*
- *Pasos en el procedimiento. Además de la manera de registrar los resultados y los incidentes de la ejecución, se debe especificar:*
  - *La secuencia necesaria de acciones para preparar la ejecución*
  - *Acciones necesarias para empezar la ejecución*
  - *Acciones necesarias durante la ejecución*
  - *Cómo se realizarán las medidas (por ejemplo, el tiempo de respuesta)*
  - *Acciones necesarias para suspender la prueba (cuando los acontecimientos no previstos lo obliguen)*
  - *Puntos para reinicio de la ejecución y acciones necesarias para el reinicio en estos puntos*
  - *Acciones necesarias para detener ordenadamente la ejecución*
  - *Acciones necesarias para restaurar el entorno y dejarlo en la situación existente antes de las pruebas*
  - *Acciones necesarias para tratar los acontecimientos anómalos*

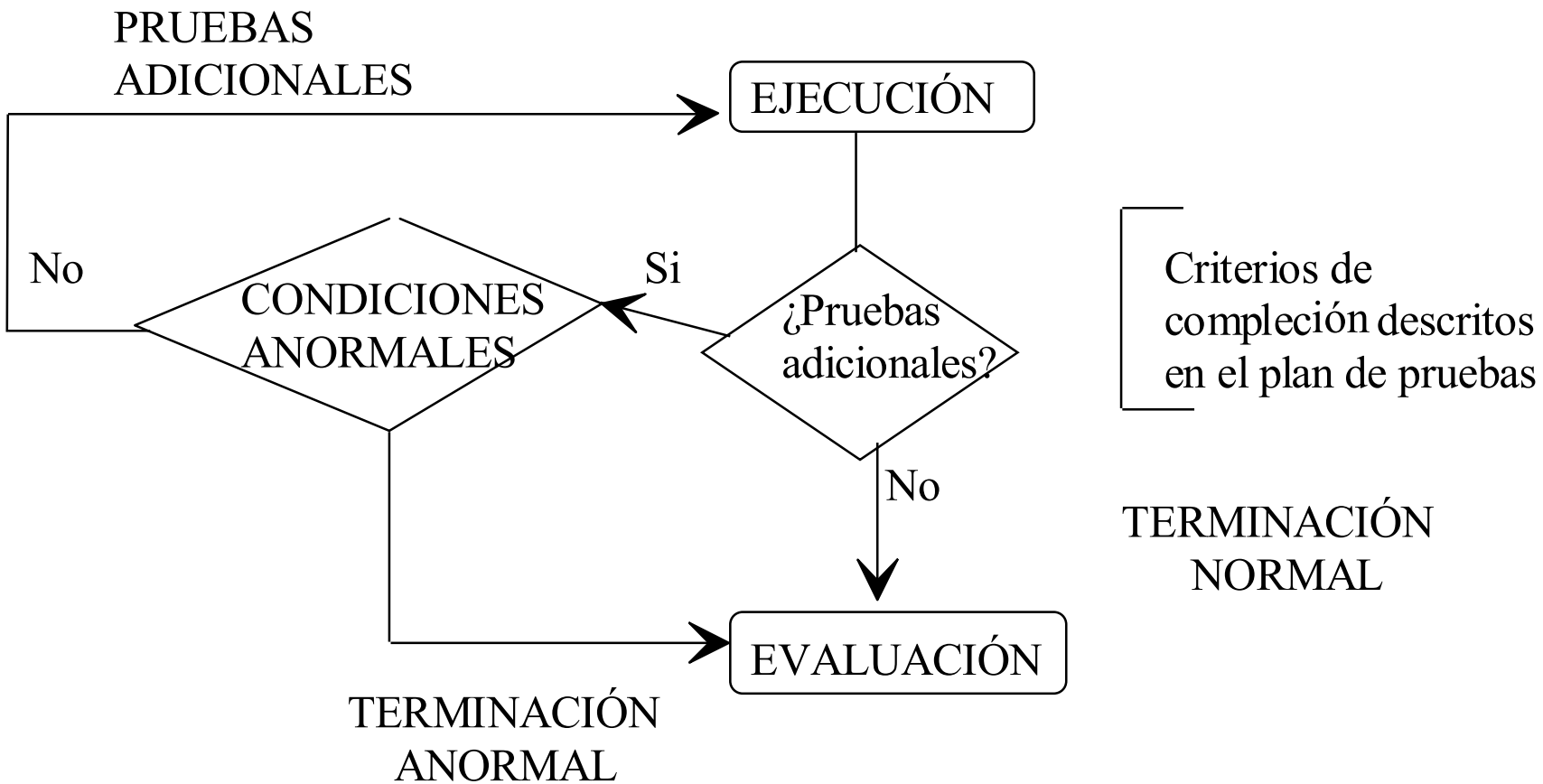
PROCESO DE PRUEBAS, TRAS EL DISEÑO DE CASOS, SEGUN EL ESTANDAR IEEE 1008



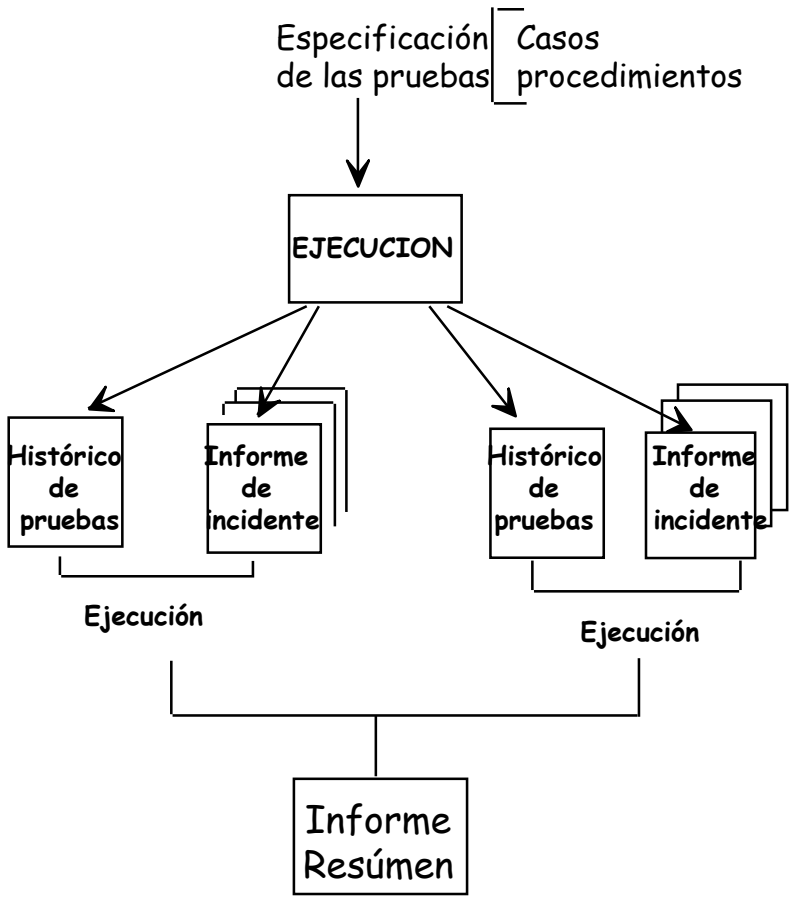
DETALLES DEL PROCESO *EJECUTAR*

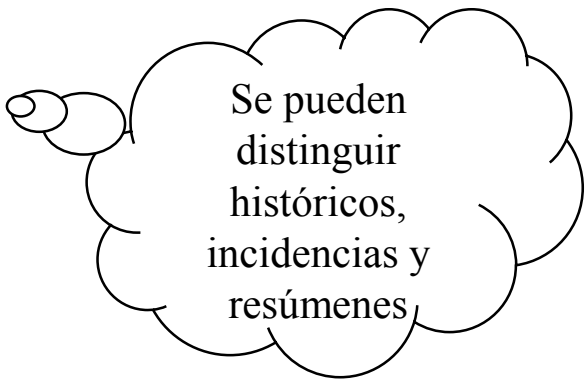


DETALLES DEL PROCESO DE COMPROBACION DE LA TERMINACION DE LAS PRUEBAS



DOCUMENTACION RELACIONADA CON LA EJECUCION DE LAS PRUEBAS SEGUN EL ESTANDAR IEEE 829





HISTORICO DE PRUEBAS

Objetivo

El histórico de pruebas (test log) documenta todos los hechos relevantes ocurridos durante la ejecución de las pruebas

HISTORICO DE PRUEBAS

Estructura fijada en el estándar:

- *Identificador*
- *Descripción de la prueba: elementos probados y entorno de la prueba*
- *Anotación de datos sobre cada hecho ocurrido (incluido el comienzo y el final de la prueba)*
  - *Fecha y hora*
  - *Identificador de informe de incidente*
- *Otras informaciones*

INFORME DE INCIDENTE

Objetivo:

El informe de incidente (test incident report) documenta cada incidente (por ejemplo, una interrupción en las pruebas debido a un corte de electricidad, bloqueo del teclado, etc.) ocurrido en la prueba y que requiera una posterior investigación.

INFORME DE INCIDENTE

Estructura fijada en el estándar:

- ✉ *Identificador*
- ✉ *Resumen del incidente*
- ✉ *Descripción de datos objetivos (fecha/hora, entradas, resultados esperados, etc)*
- ✉ *Impacto que tendrá sobre las pruebas*

INFORME RESUMEN DE PRUEBAS

Objetivo:

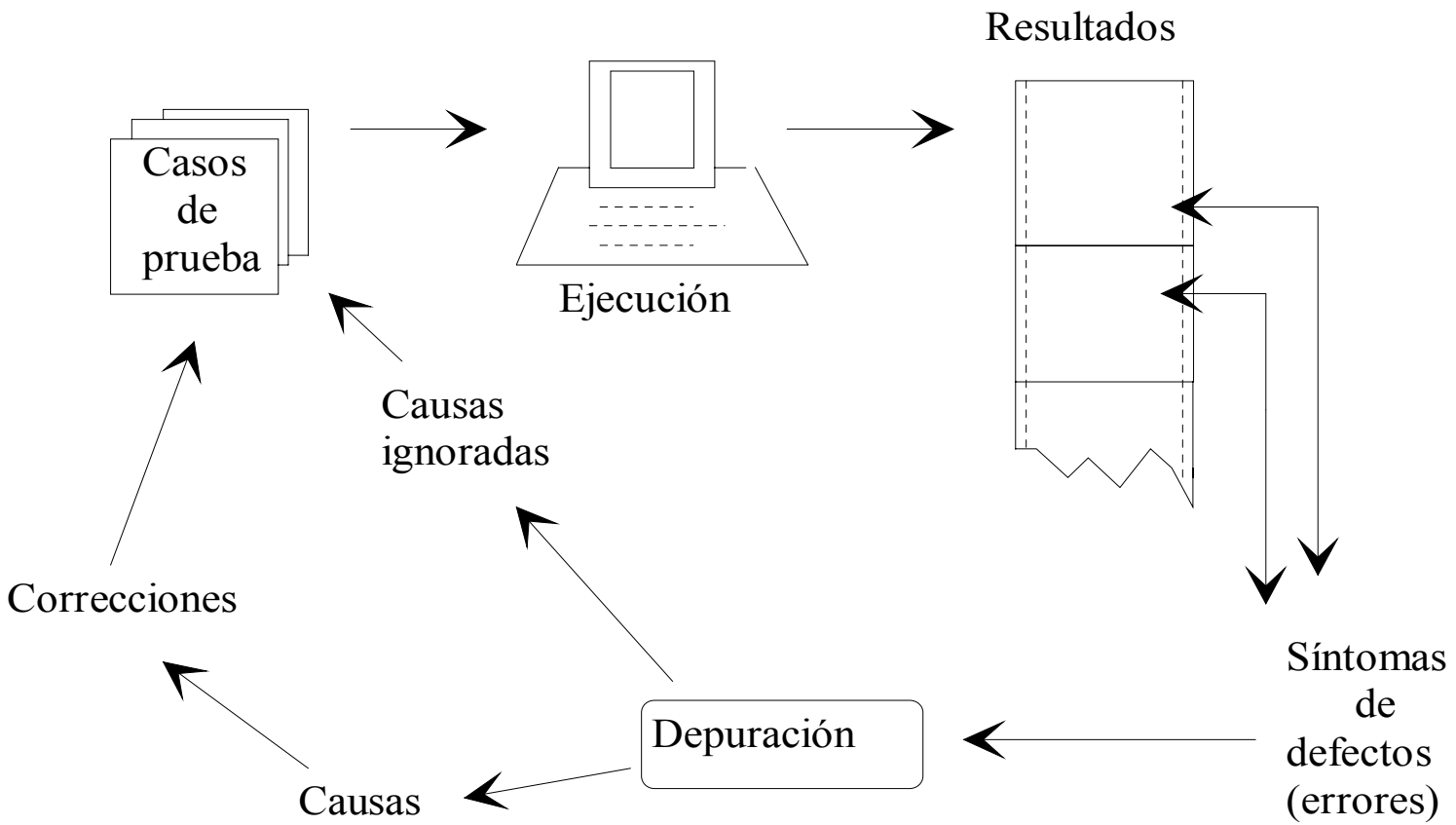
El informe resumen (test summary report) resume los resultados de las actividades de prueba (las señaladas en el propio informe) y aporta una evaluación del software basada en dichos resultados

INFORME RESUMEN DE LAS PRUEBAS

Estructura fijada en el estándar:

- ✉ *Identificador*
- ✉ *Resumen de la evaluación de los elementos probados*
- ✉ *Variaciones del software respecto a su especificación de diseño, así como las variaciones en las pruebas*
- ✉ *Valoración de la extensión de la prueba (cobertura lógica, funcional, de requisitos, etc.)*
- ✉ *Resumen de los resultados obtenidos en las pruebas*
- ✉ *Evaluación de cada elemento software sometido a prueba (evaluación general del software incluyendo las limitaciones del mismo)*
- ✉ *Firmas y aprobaciones de quienes deban supervisar el informe*

RELACION ENTRE LAS PRUEBAS Y LA DEPURACION



Depuración: el proceso de analizar y corregir los defectos que se sospecha que contiene el software

CONSEJOS PARA LA DEPURACION

Localización del error

- ✓ *Analizar la información y pensar (analizar bien, en lugar de aplicar un enfoque aleatorio de búsqueda del defecto)*
- ✓ *Al llegar a un punto muerto, pasar a otra cosa (refresca la mente)*
- ✓ *Al llegar a un punto muerto, describir el problema a otra persona (el simple hecho de describir el problema a alguien puede ayudar)*
- ✓ *Usar herramientas de depuración sólo como recurso secundario (no sustituir el análisis mental)*
- ✓ *No experimentar cambiando el programa (no sé qué está mal, así que cambiaré esto y veré lo que sucede → NO)*
- ✓ *Se deben atacar los errores individualmente*
- ✓ *Se debe fijar la atención también en los datos (no sólo en la lógica del programa)*

CONSEJOS PARA LA DEPURACION

Corrección del error

- ✓ *Donde hay un defecto, suele haber más (lo dice la experiencia)*
- ✓ *Debe fijarse el defecto, no sus síntomas (no debemos enmascarar síntomas, sino corregir el defecto)*
- ✓ *La probabilidad de corregir perfectamente un defecto no es del 100% (cuando se corrige, hay que probarlo)*
- ✓ *Cuidado con crear nuevos defectos (al corregir defectos se producen otros nuevos)*
- ✓ *La corrección debe situarnos temporalmente en la fase de diseño (hay que retocar desde el comienzo, no sólo el código)*
- ✓ *Cambiar el código fuente, no el código objeto*

ANALISIS DE ERRORES O ANALISIS CAUSAL

El objetivo del análisis causal es proporcionar información sobre la naturaleza de los defectos. Para ello es fundamental recoger para cada defecto detectado esta información:

¿Cuándo se cometió?

¿Quién lo hizo

¿Qué se hizo mal?

¿Cómo se podría haber prevenido?

¿Por qué no se detectó antes?

¿Cómo se podría haber detectado antes?

¿Cómo se encontró el error?

Esta información no debería ser usada para evaluar al personal, sino para la formación del mismo sobre cómo prevenir los errores. También se utiliza para predecir futuros fallos software

ESTRATEGIA DE APLICACIÓN DE LAS PRUEBAS

☑ *Se analiza cómo plantear las pruebas en el Ciclo de Vida.*

☑ *La estrategia de pruebas suele seguir estas etapas*

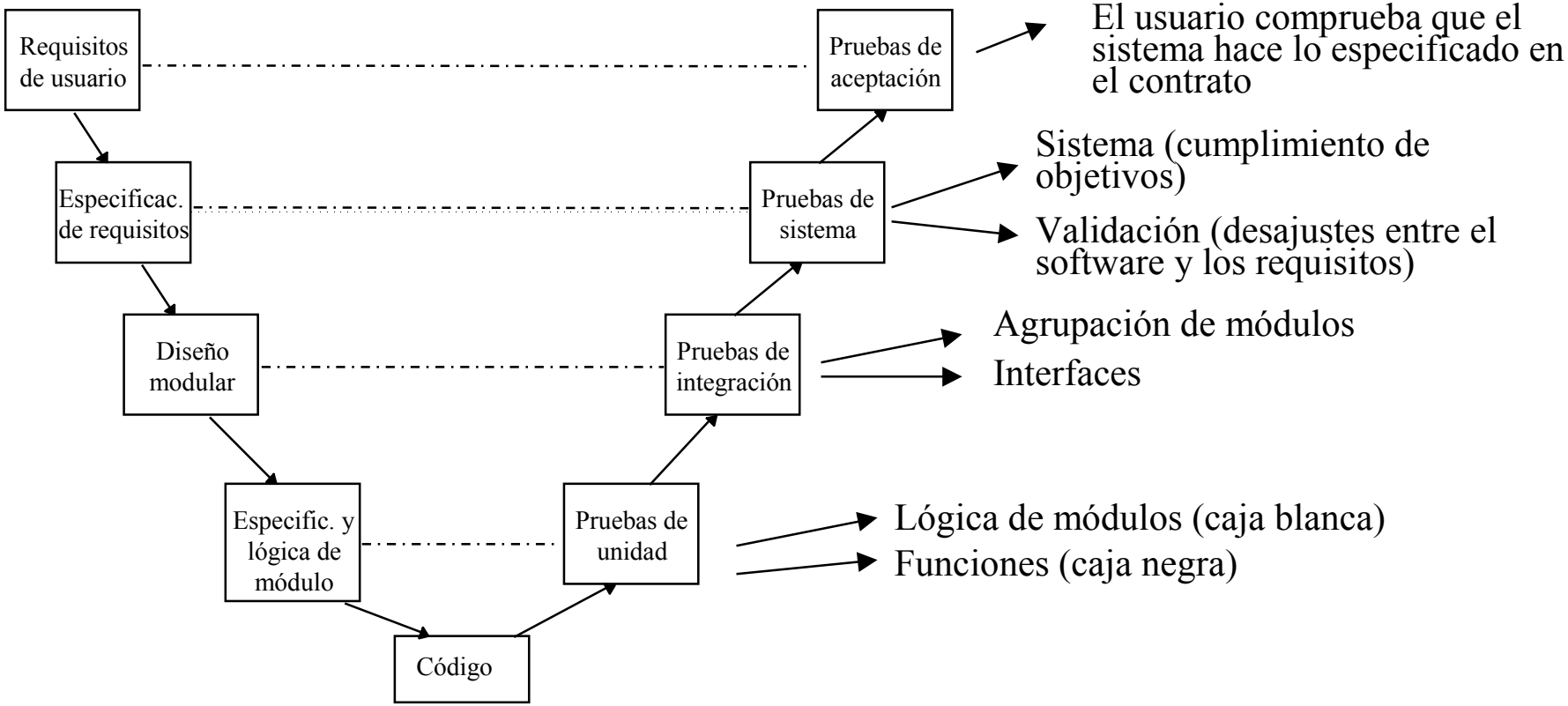
- *Comenzar pruebas a nivel de módulo*
- *Continuar hacia la integración del sistema completo y a su instalación*
- *Culminar con la aceptación del producto por parte del cliente*

ESTRATEGIA DE APLICACIÓN DE LAS PRUEBAS

Etapas típicas más en detalle

- ☑ *Se comienza en la prueba de cada módulo, que normalmente la realiza el propio personal de desarrollo en su entorno*
- ☑ *Con el esquema del diseño del software, los módulos probados se integran para comprobar sus interfaces en el trabajo conjunto (prueba de integración)*
- ☑ *El software totalmente ensamblado se prueba como un conjunto para comprobar si cumple o no tanto los requisitos funcionales como los requisitos de rendimientos, seguridad, etc. (prueba funcional o de validación).*
- ☑ *El software ya validado se integra con el resto del sistema (por ejemplo, elementos mecánicos, interfaces electrónicas, etc.) para probar su funcionamiento conjunto (prueba del sistema)*
- ☑ *Por último, el producto final se pasa a la prueba de aceptación para que el usuario compruebe en su propio entorno de explotación si lo acepta como está o no (prueba de aceptación)*

RELACION ENTRE PRODUCTOS DE DESARROLLO Y NIVELES DE PRUEBA



Existe una correspondencia entre cada nivel de prueba y el trabajo realizado en cada etapa del desarrollo

PRUEBA DE UNIDAD

Se trata de las pruebas formales que permiten declarar que un módulo está listo y terminado (no las informales que se realizan mientras se desarrollan los módulos)

Hablamos de una **unidad de prueba** para referirnos a uno o más módulos que cumplen las siguientes condiciones [IEEE, 1986a]:

- Todos son del mismo programa
- Al menos uno de ellos no ha sido probado
- El conjunto de módulos es el objeto de un proceso de prueba

La prueba de unidad puede abarcar desde un módulo hasta un grupo de módulos (incluso un programa completo)

Estas pruebas suelen realizarlas el propio personal de desarrollo, pero evitando que sea el propio programador del módulo

PRUEBAS DE INTEGRACION

Implican una progresión ordenada de pruebas que van desde los componentes o módulos y que culminan en el sistema completo

El orden de integración elegido afecta a diversos factores, como lo siguientes:

- La forma de preparar casos
- Las herramientas necesarias
- El orden de codificar y probar los módulos
- El coste de la depuración
- El coste de preparación de casos

PRUEBAS DE INTEGRACION

Tipos fundamentales de integración

- Integración incremental. Se combina el siguiente módulo que se debe probar con el conjunto de módulos que ya han sido probados
  - **ascendente. Se comienza por los módulos hoja.**
  - **descendente. Se comienza por el módulo raíz.**
- Integración no incremental. Se prueba cada módulo por separado y luego se integran todos de una vez y se prueba el programa completo

Habitualmente las pruebas de unidad y de integración se solapan y mezclan en el tiempo.

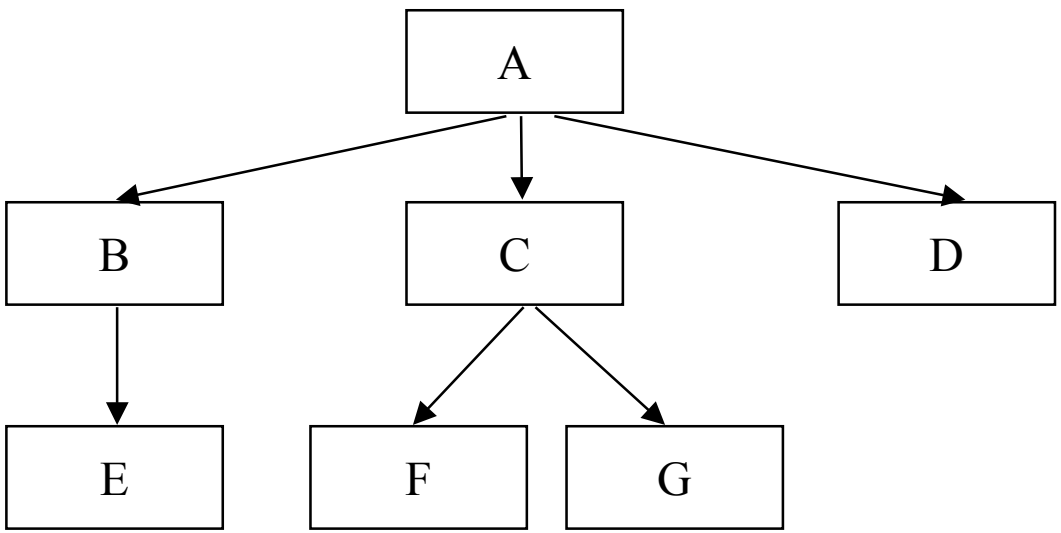
INTEGRACIÓN INCREMENTAL ASCENDENTE

El proceso es el siguiente

- Se combinan los módulos de bajo nivel en grupos que realicen una función o subfunción específica (o quizás si no es necesario, individualmente) → de este modo reducimos el número de pasos de integración.
- Se escribe para cada grupo un módulo impulsor o conductor → de este modo permitimos simular la llamada a los módulos, introducir datos de prueba y recoger resultados.
- Se prueba cada grupo mediante su impulsor
- Se eliminan los módulos impulsores y se sustituyen por los módulos de nivel superior en la jerarquía.

DISEÑO MODULAR SOBRE EL QUE SE REALIZA LA INTEGRACION

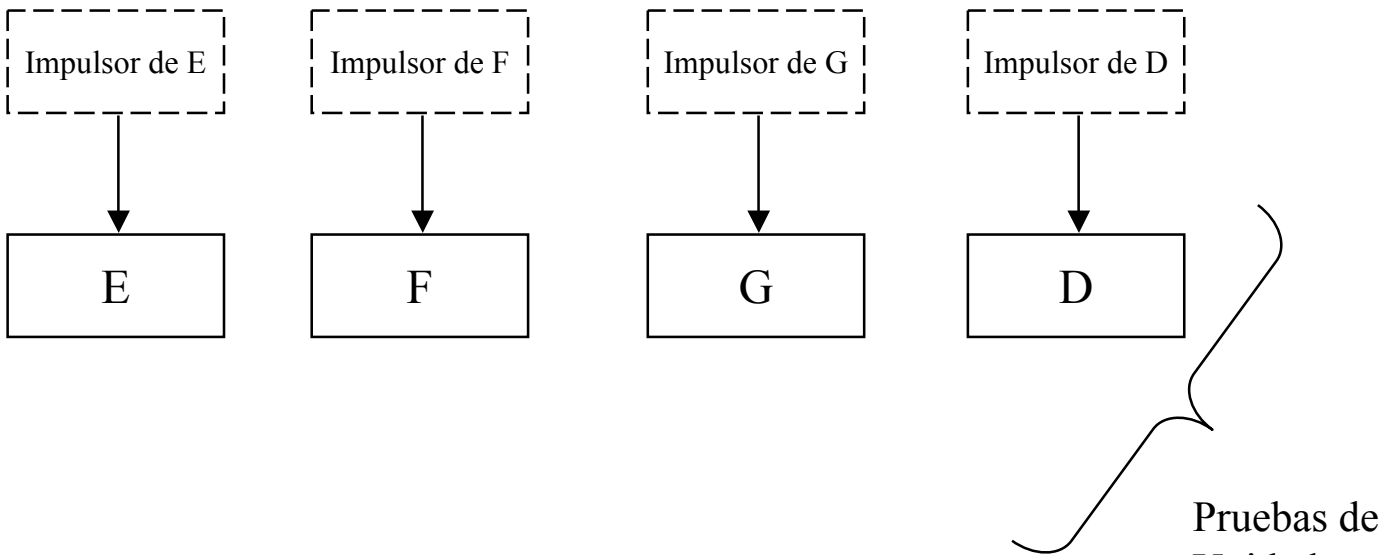
Supongamos esta estructura de programa



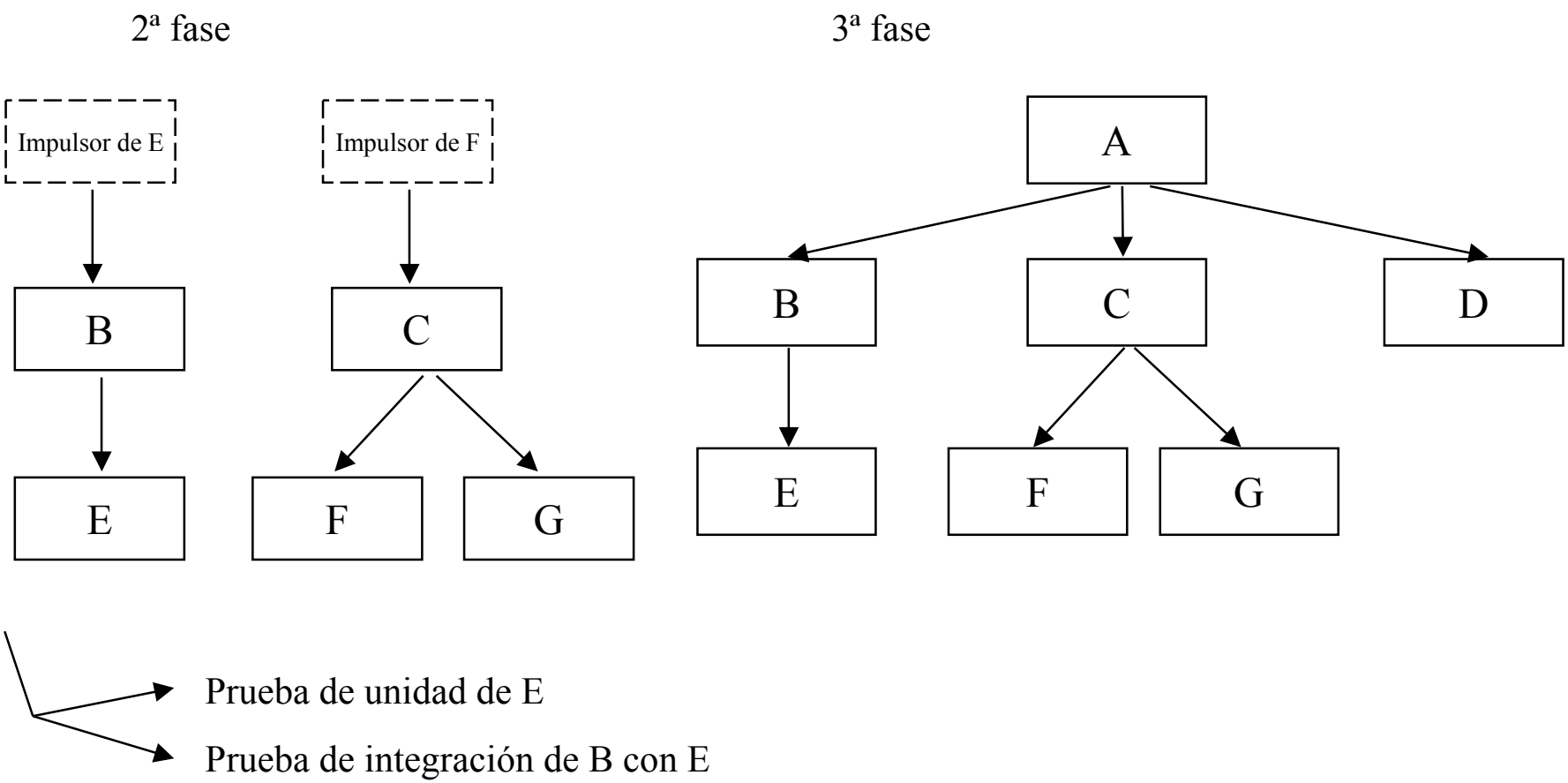
PRIMERA FASE DE LA INTEGRACION ASCENDENTE DEL EJEMPLO

1ª fase

Se definen los impulsores para nodos hoja y se ejecutan



SEGUNDA Y TERCERA FASE DE LA INTEGRACION ASCENDENTE DEL EJEMPLO



INTEGRACIÓN INCREMENTAL DESCENDENTE

Comienza por el módulo de control principal y va incorporando módulos subordinados progresivamente.

No hay un orden adecuado de integración, pero unos consejos son los siguientes:

-Si hay secciones críticas (especialmente complejas) se deben integrar lo antes posible.
-El orden de integración debe incorporar cuanto antes los módulos de entrada/salida para facilitar la ejecución de puebas

Existen dos formas básicas de hacer esta integración:
-Primero en profundidad: Se van completando ramas del árbol (A B E C F G D)
-Primero en anchura: Se van completando niveles de jerarquía (A B C D E F G)

ETAPAS FUNDAMENTALES DE LA INTEGRACION DESCENDENTE

- El módulo raíz es el primero: Se escriben módulos ficticios que simulan los subordinados
- Una vez probado el módulo raíz (sin detectarse ya ningún defecto), se sustituye uno de los subordinados ficticios por el módulo correspondiente según el orden elegido
- Se ejecutan las correspondientes pruebas cada vez que se incorpora un módulo nuevo
- Al terminar cada prueba, se sustituye un ficticio por su correspondiente real
- Conviene repetir algunos casos de prueba de ejecuciones anteriores para asegurarse de que no se ha introducido ningún defecto nuevo

MÓDULOS FICTICIOS

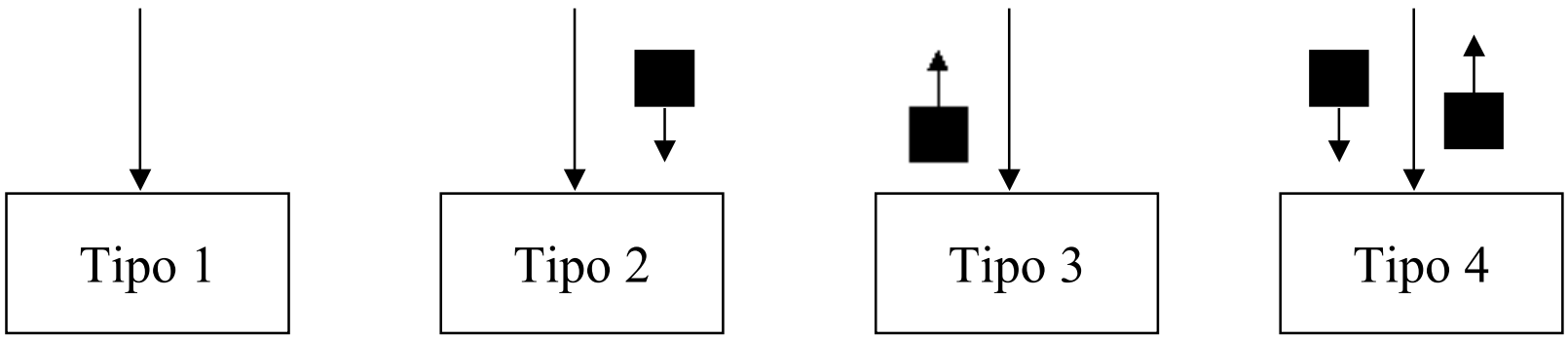
La creación de módulos ficticios subordinados es más complicada que la creación de impulsores

-Complejidad

+Complejidad

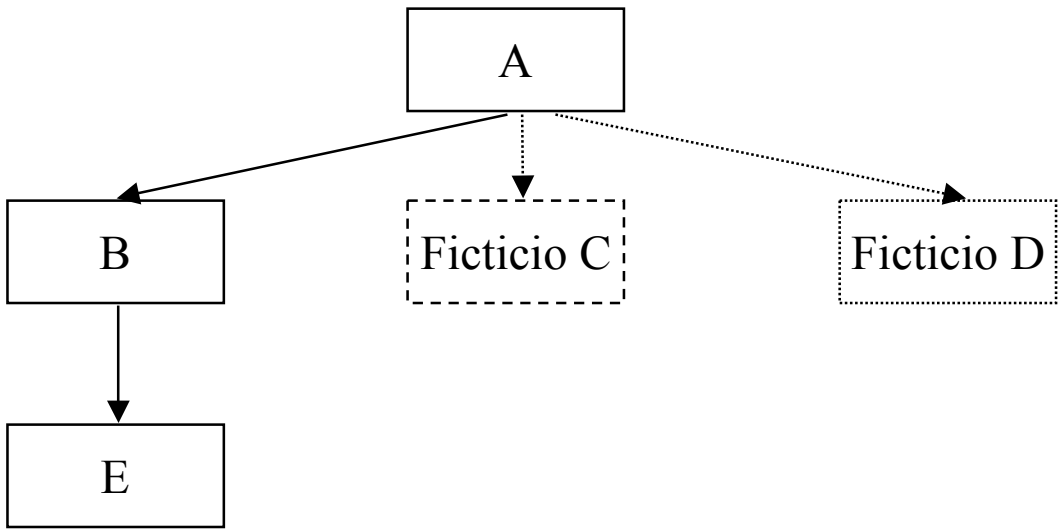
- Módulos que sólo muestran un mensaje de traza
- Módulos que muestran los parámetros que se les pasa
- Módulos que devuelven un valor que no depende de los parámetros que se pasen como entrada
- Módulos que, en función de los parámetros pasados, devuelven un valor de salida que más o menos se corresponda con dicha entrada

UNA POSIBLE CLASIFICACION DE LOS MODULOS FICTICIOS



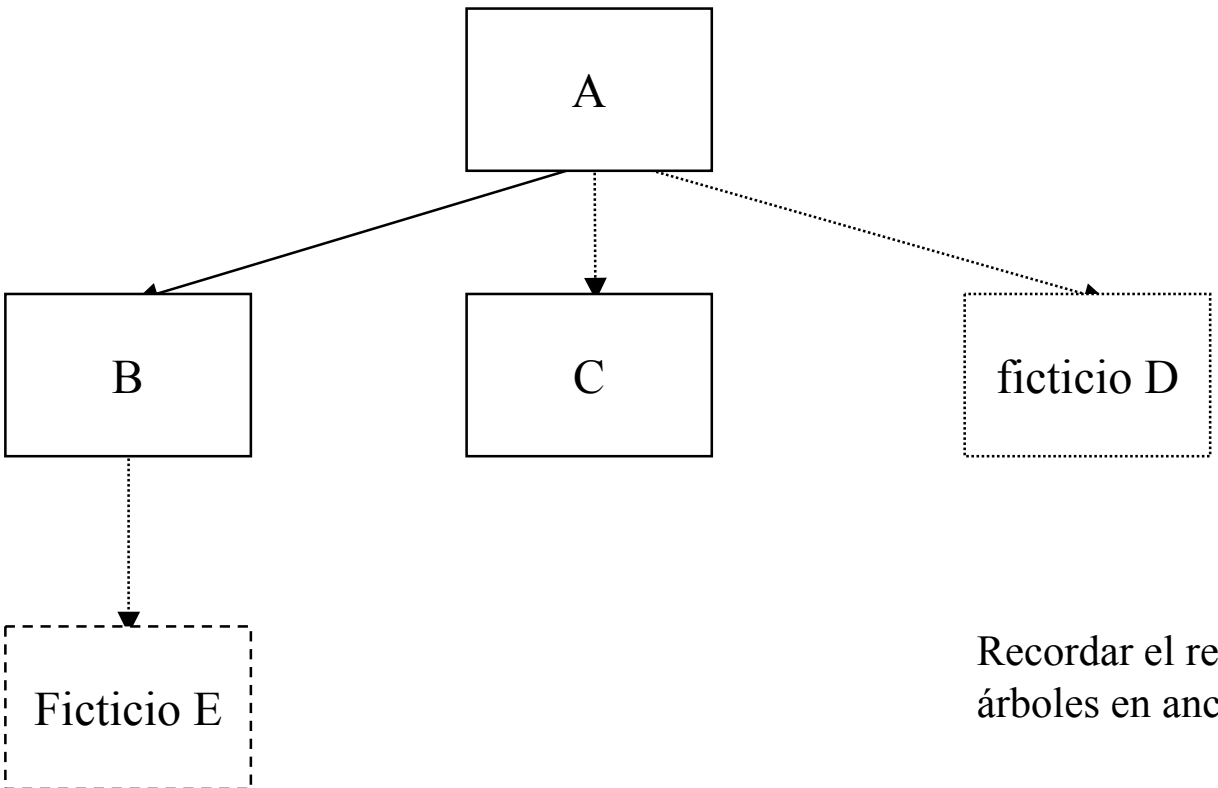
Tipo 1	Tipo 2	Tipo 3	Tipo 4
Muestra mensaje	Muestra param. de entrada	Devuelve valor	Recibe y devuelve valor

UN PASO INTERMEDIO EN LA INTEGRACION DESCENDENTE PRIMERO EN LA PROFUNDIDAD DEL EJEMPLO



Recordar el recorrido de árboles en profundidad

UN PASO INTERMEDIO EN LA INTEGRACION DESCENDENTE PRIMERO EN LA ANCHURA DEL EJEMPLO



Recordar el recorrido de árboles en anchura

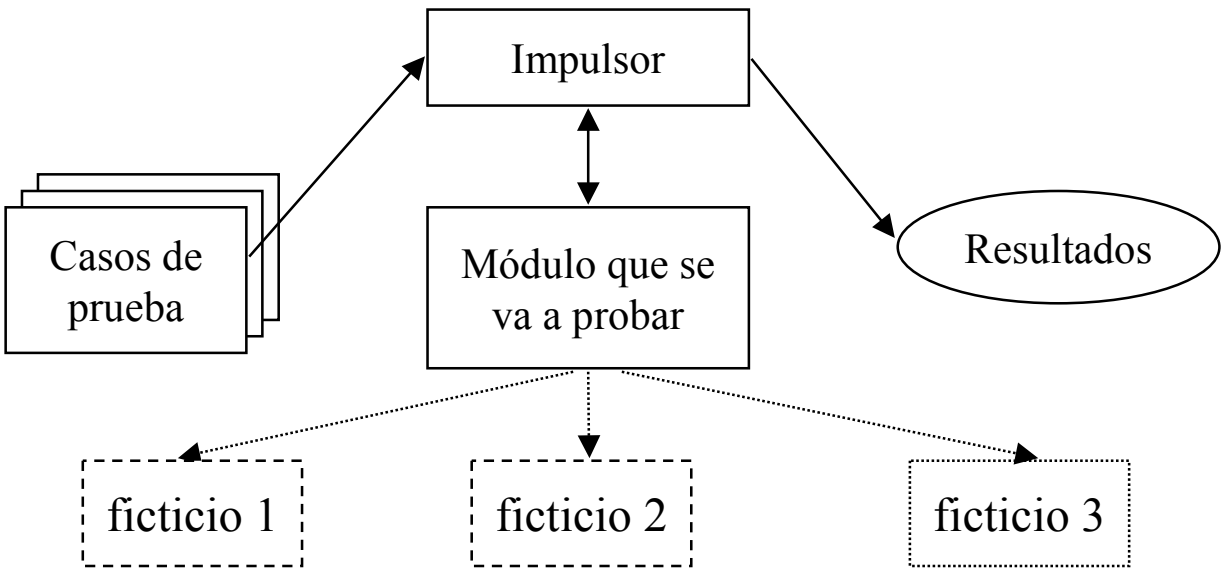
INTEGRACION NO INCREMENTAL

Cada módulo que tiene que ser probado necesita lo siguiente:

- Un módulo impulsor, que transmite o «impulsa» los datos de prueba al módulo y muestra los resultados de dichos casos de prueba
- Uno o más módulos ficticios que simulan la función de cada módulo subordinado llamado por el módulo que se va a probar

Una vez probado cada módulo por separado, se ensamblan todos de una vez y se prueban en conjunto

PRUEBA TIPICA DEL MODULO EN LA INTEGRACION NO INCREMENTAL



COMPARACION DE LOS DISTINTOS TIPOS DE INTEGRACION

*Ventajas de la **no incremental:***

- Requiere menos tiempo de máquina para las pruebas, ya que se prueba de una sola vez la combinación de los módulos
- Existen más oportunidades de probar módulos en paralelo

*Ventajas de la **incremental:***

- Requiere menos trabajo, ya que se escriben menos módulos impulsores y ficticios
- Los defectos y errores en las interfaces se detectan antes, ya que se empieza antes a probar las uniones entre los módulos
- La depuración es mucho más fácil, ya que si se detectan los síntomas de un defecto en un paso de la integración hay que atribuirlo muy probablemente al último módulo incorporado
- Se examina con mayor detalle el programa, al ir comprobando cada interfaz poco a poco

VENTAJAS Y DESVENTAJAS DE LAS INTEGRACIONES ASCENDENTE Y DESCENDENTE

Ascendente	Descendente
Ventajas • Es un método ventajoso si aparecen grandes fallos en la parte inferior del programa, ya que se prueba antes. • La entradas para las pruebas son más fáciles de crear, puesto que los módulos inferiores tienen funciones más específicas. • Es más fácil observar los resultados de la prueba, ya que es en los módulos inferiores donde se elaboran los datos (los superiores suelen ser módulos de control).	*Ventajas* • Es ventajosa si aparecen grandes defectos en los niveles superiores del programa, ya que se prueban antes. • Una vez incorporadas las funciones de entrada/salida, es fácil manejar los casos de prueba. • Permite ver antes una estructura previa del programa, lo que facilita el hacer demostraciones y ayuda a mantener la moral.
Desventajas • Se requieren módulos impulsores, que deben codificarse. • El programa, como entidad, sólo aparece cuando se agrega el último módulo.	*Desventajas* • Se requieren módulos ficticios que suelen ser complejos de crear. • Antes de incorporar la entrada/salida resulta complicado el manejo de los casos de prueba. • Las entradas para las pruebas pueden ser difíciles o imposibles de crear, puesto que, a menudo, se carece de los módulos inferiores que proporcionan los detalles de operación. • Es más difícil observar la salida, ya que los resultados surgen de los módulos inferiores. • Pueden inducir a diferir la terminación de la prueba de ciertos módulos.

PRUEBA DEL SISTEMA

Es el proceso de prueba de un sistema integrado de hardware y software para comprobar lo siguiente:

- Cumplimiento de todos los requisitos funcionales, considerando el producto software final al completo en un entorno de sistema
- El funcionamiento y rendimiento en las interfaces hardware, software, de usuario y de operador
- Adecuación de la documentación de usuario
- Ejecución y rendimiento en condiciones límite y de sobrecarga

FUENTES DE DISEÑO DE CASOS DE PRUEBA DEL SISTEMA

- Casos basados en los requisitos gracias a técnicas de caja negra aplicadas a las especificaciones

- Casos necesarios para probar el rendimiento del sistema y de su capacidad funcional (pruebas de volumen de datos, de límites de procesamiento, etc.). Este tipo de pruebas suelen llamarse pruebas de sobrecarga (*stress testing*)

- Casos basados en el diseño de alto nivel aplicando técnicas de caja blanca a los flujos de datos de alto nivel (por ejemplo, de los diagramas de flujo de datos)

PRUEBA DE ACEPTACION

Es la prueba planificada y organizada formalmente para determinar si se cumplen los requisitos de aceptación marcados por el cliente.

Sus características principales son las siguientes:

- Participación del usuario
- Está enfocada hacia la prueba de los requisitos de usuario especificados.
- Está considerada como la fase final del proceso para crear una confianza en que el producto es el apropiado para su uso en explotación

RECOMENDACIONES GENERALES

- Debe contarse con criterios de aceptación previamente aprobados por el usuario

- No hay que olvidar validar también la documentación y los procedimientos de uso (por ejemplo,mediante una auditoría)

- Las pruebas se deben realizar en el entorno en el que se utilizará el sistema (lo que incluye al personal que lo maneja)