



Universidad Nacional de Tucumán



Universidad Nacional de Tucumán
Facultad de Ciencias Exactas y Tecnología

Ingeniería de Software II

Pruebas del Software

Mg. Héctor A. Valdecantos

Minuto a minuto cuando codeamos...

¿Cómo sabemos que el código es correcto?

¿Cómo sabemos qué es lo que el cliente pidió?

¿Cómo sabemos si hemos terminado?

Técnicas de Prueba del Software

- Se basan en cómo escribir casos de prueba efectivos:
 - Dominio del problema vs. tecnología
 - Según Myers es un Arte [1]
 - Sentido destructivo
- Las partes reusables reducen el esfuerzo puesto en las pruebas
 - Las partes reusables ya están testeadas!
- Un testeo completo es imposible y, en general, es poco práctico

[1][G. Myers, El arte de las pruebas de Software]

Objetivos de las pruebas del Software

- Descubrir errores de un proceso en ejecución (run-time)
- Generar casos de prueba que tengan una alta probabilidad de descubrir errores
- Una prueba tiene éxito si descubre un error no descubierto anteriormente

[G. Myers, El arte de las pruebas de Software]

La prueba no puede asegurar la ausencia de errores, sólo puede demostrar que existen defectos en el software.

¿Qué casos de prueba son necesarios para un programa que recibe tres enteros que representan el largo de los lados de un triángulo y muestra un mensaje diciendo si es un triángulo es escaleno, isósceles, o equilátero?

1. ¿Tiene un caso de prueba que represente un triángulo escaleno válido? Tenga en cuenta que los casos de prueba como 1, 2, 3 y 2, 5, 10 no garantizan un sí, porque un triángulo que tiene estas dimensiones no es válido. **Teorema:** desigualdad triangular tal que $(b+c>a \ \& \ a+c>b \ \& \ a+b>c)$
2. ¿Tiene un caso de prueba que representa un triángulo equilátero válido?
3. ¿Tiene un caso de prueba que representa un triángulo isósceles válido? (Tenga en cuenta que un caso de prueba que representa 2, 2, 4 no contaría porque no es un triángulo válido.)
4. ¿Tiene al menos tres casos de prueba que representan isósceles válidos? Triángulos tales que pruebe las tres permutaciones de dos lados iguales (como, 3, 3, 4; 3, 4, 3 y 4, 3, 3)?
5. ¿Tiene un caso de prueba en el que un lado tiene un valor cero?
6. ¿Tiene un caso de prueba en el que un lado tiene un valor negativo?
7. ¿Tiene un caso de prueba con tres enteros mayores que cero, de modo que la suma de dos de los números es igual al tercero? (Es decir, si el programa dijo que 1, 2, 3 representa un triángulo escaleno, contendría un bug.)
8. ¿Tiene al menos tres casos de prueba en la categoría 7 de manera que tenga probado las tres permutaciones donde la longitud de un lado es igual a la suma de las longitudes de los otros dos lados (p. ej., 1, 2, 3; 1, 3, 2 y 3, 1, 2)?
9. ¿Tiene un caso de prueba con tres enteros mayores que cero tal que la suma de dos de los números es menor que la tercera (como 1, 2, 4 o 12, 15, 30)?
10. ¿Tiene al menos tres casos de prueba en la categoría 9 de modo que tenga probado las tres permutaciones (por ejemplo, 1, 2, 4, 1, 4, 2 y 4, 1, 2)?
11. ¿Tiene un caso de prueba en el que todos los lados son cero (0, 0, 0)?
12. ¿Tiene al menos un caso de prueba que especifique valores no enteros? (cómo 2.5, 3.5, 5.5)?
13. ¿Tiene al menos un caso de prueba que especifique el número incorrecto de valores (dos en lugar de tres enteros, por ejemplo)?
14. Para cada caso de prueba, especificó el resultado esperado del programa además de los valores de entrada?

Qué es la prueba de SW

Fuentes de problemas



requerimientos

análisis

restricciones

tecnología

recursos: memoria,

disco, process, IO.

lenguajes:

diseño

implementación

pruebas

Qué es la prueba de SW

“La prueba es el proceso de demostrar que no hay errores presentes”

“El propósito de las pruebas es mostrar que un programa realiza sus funciones correctamente”

“Las pruebas son el proceso de establecer la confianza de que un programa hace lo que se supone que debe hacer”

Pero en realidad

“Las pruebas es un proceso de ejecutar un programa con la intención de encontrar y corregir errores”


¿Cuando una ‘test suite’ es exitosa?

El propósito de la prueba de SW

**si descubre
errores**

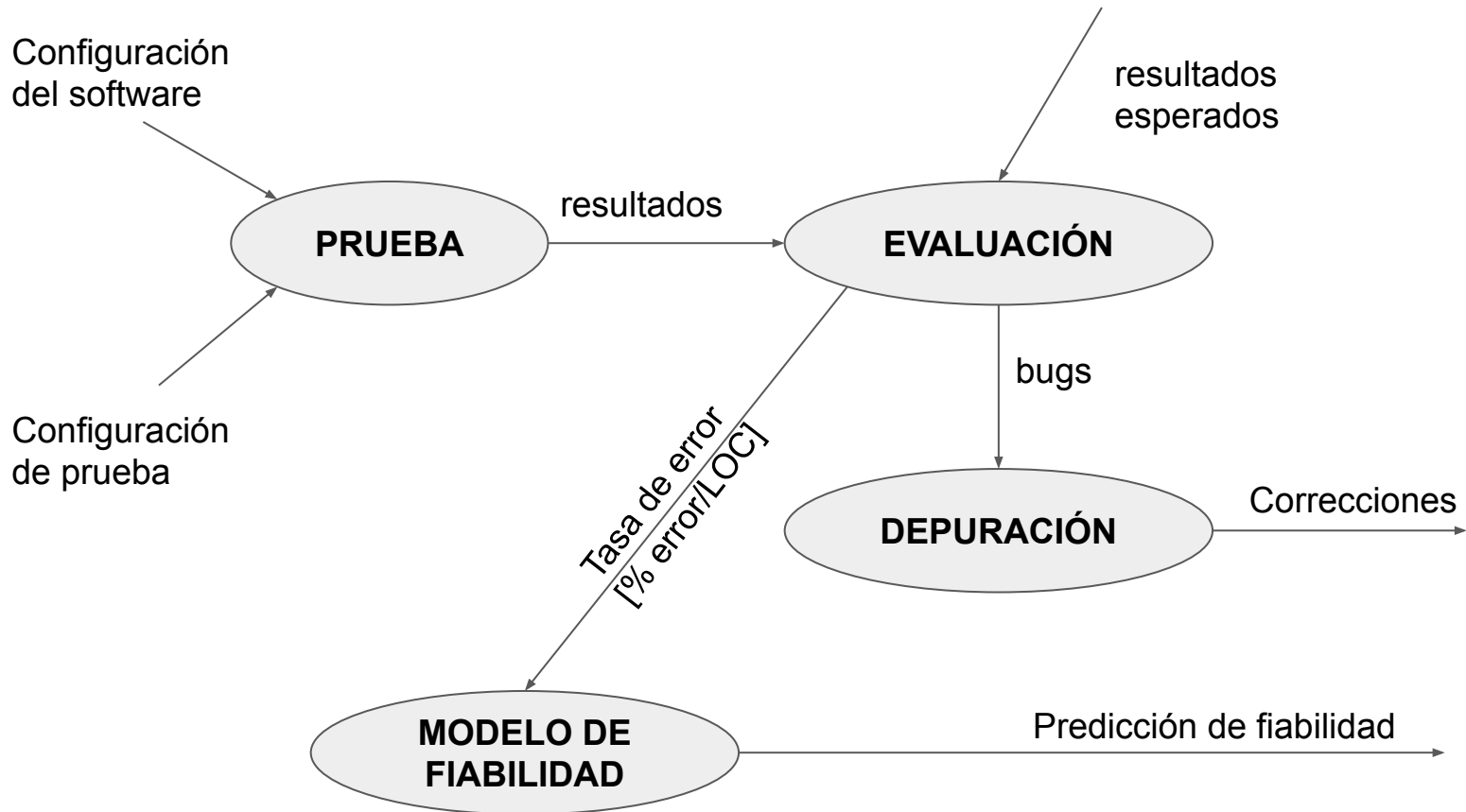


La prueba agrega valor al desarrollo de dos formas:

- Al descubrir errores en un sistema de software.
 - Estableciendo un grado de confianza en las funciones del sistema.
- 

**si no descubre
errores**

Flujo de información de la prueba



Diseño de los casos de prueba

- Pruebas de caja negra

Son pruebas sobre la interfaz del Software para demostrar que sus funciones son operativas, que la entrada es correcta y la salida es correcta. No tiene en cuenta la estructura interna del Software. (aka IO-driven testing)

- Pruebas de caja blanca

Aseguran que la operación interna se ajusta a las especificaciones. Prueba los detalles internos: mecanismos lógicos, se ejercitan los lazos y condiciones. Se examina el estado de cada programa en varios puntos.

Pruebas de caja negra

Se centran en requisitos funcionales (especificaciones) que a través de un conjunto de entradas se ejercitan las diferentes funciones del SW. Enfoque complementario para descubrir errores como:

- Funciones incorrectas o ausentes
- Errores de interfaz
- Errores de estructuras de datos o en acceso a BD
- Errores de rendimiento
- Errores de inicialización o finalización (pre y post condition)

Estas pruebas se aplican en etapas posteriores a las pruebas de caja blanca. Se podrían considerar como pruebas a “**gran escala**”

Pruebas de caja blanca

Se derivan casos de prueba a partir del estudio de la estructura de control del diseño procedimental que:

- Garanticen se prueba por lo menos una vez todos los caminos independientes
- Se ejercitan todas las condiciones lógicas (**true|false**)
- Se ejecuten todos los lazos en sus límites
- Se ejerciten las estructuras internas de datos para asegurar su validez

Con estas pruebas se descubren errores que muchas veces pasan inadvertidos en las pruebas de caja negra.

Se consideran pruebas a “**pequeña escala**”

¿Si se quiere encontrar la mayor cantidad de errores posibles, qué criterio usaría para los diferentes enfoques vistos? (para un enfoque de caja negra o de caja blanca)

- Para el enfoque de caja negra se usa el criterio de testeo exhaustivo de la entrada
- Para el enfoque de caja blanca el criterio de testeo exhaustivo de caminos

Implicaciones de las pruebas

- No se puede probar un programa y garantizar que está libre de errores.
- Una consideración fundamental en las pruebas, es el de economía de las pruebas.

Características de un buen test

- Alta probabilidad de encontrar un error
- No es redundante
- Tiene que ser lo mejor de su categoría (“best of breed”)
- No tiene que ser muy simple
- No tiene que ser muy complejo

Atributos de calidad deseada en los tests

- Operabilidad
- Observabilidad
- Controlabilidad
- Descomponibilidad
- Simplicidad
- Estabilidad
- Comprensibilidad

[Pressman, Software Engineering: a practitioner's approach, 7th edition, pag. 483]

Estrategia de prueba de software

- La prueba es un conjunto de actividades que se deben planificar y llevar a cabo sistemáticamente.
- Una estrategia de prueba debe incluir
 - planificación de la prueba
 - el diseño de los casos de prueba
 - la ejecución de la prueba
 - la evaluación de los datos resultantes.

Estrategia de prueba de software

Características generales de plantillas de pruebas:

- Comienzan a nivel componente y trabajan hacia afuera para integrar el resto del sistema.
- Diferentes técnicas de prueba son adecuadas en diferentes momentos.
- Algunas pruebas son llevadas a cabo por el desarrollador
- Pruebas de mayor escala suelen ser llevadas por un grupo de prueba independiente (conflicto de interés)
- Testing & debugging son actividades diferentes, pero la depuración debe incluirse en cualquier estrategia de prueba.

Verificación y validación (V&V)

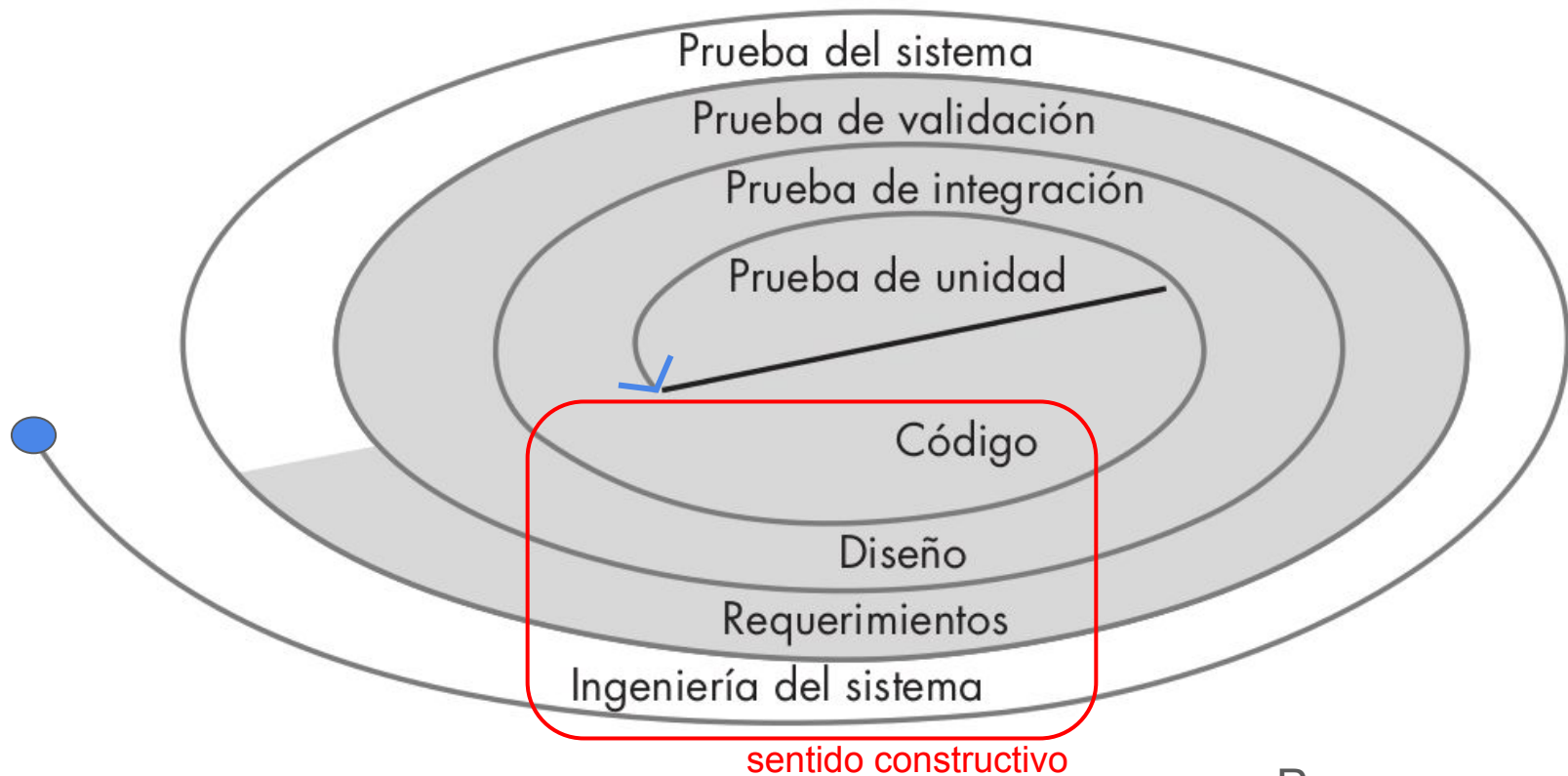
Es un concepto más amplio que las pruebas, incluye también: revisiones, auditorías, monitoreo, simulaciones, estudios, análisis...

- **Verificación:** ¿Es correcto el producto?
- **Validación:** ¿Es el producto correcto?

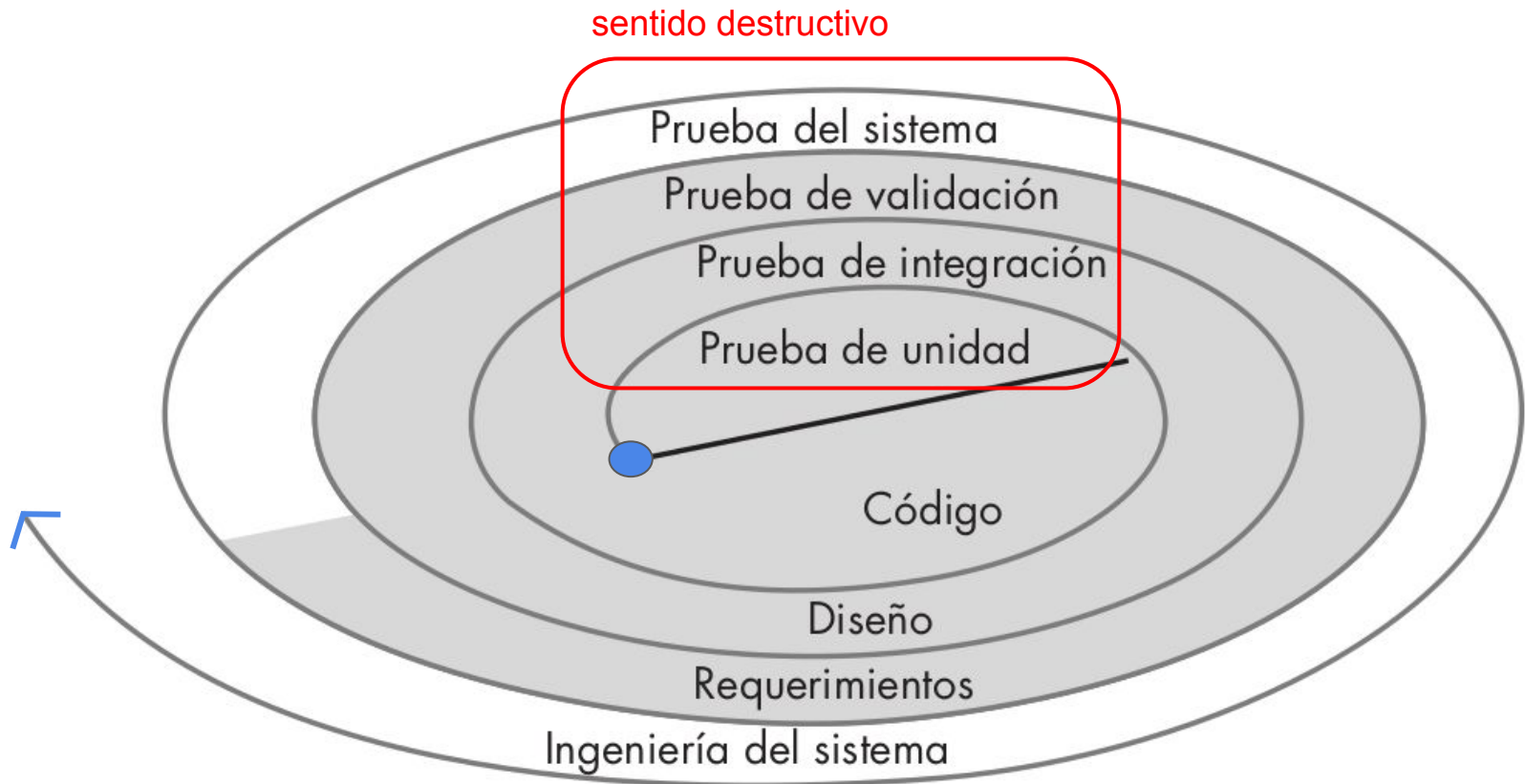
La prueba (por sí misma) no genera calidad, sólo evalúa calidad.

La calidad se incorpora al software durante el proceso de ingeniería, cuando se aplican adecuadamente los métodos y herramientas y se los someten a revisiones.

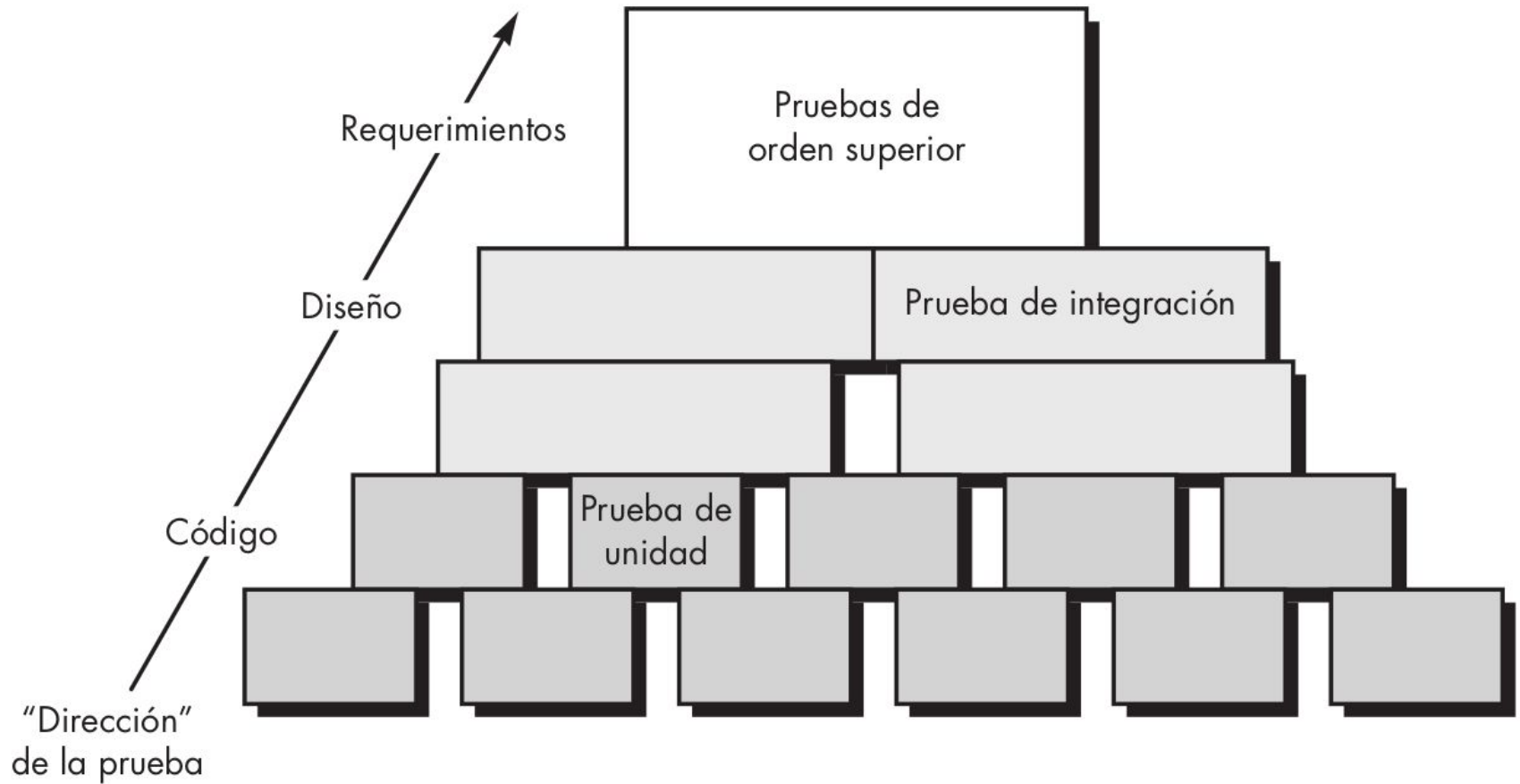
Estrategia de prueba de software (visión general)



Estrategia de prueba de software (visión general)



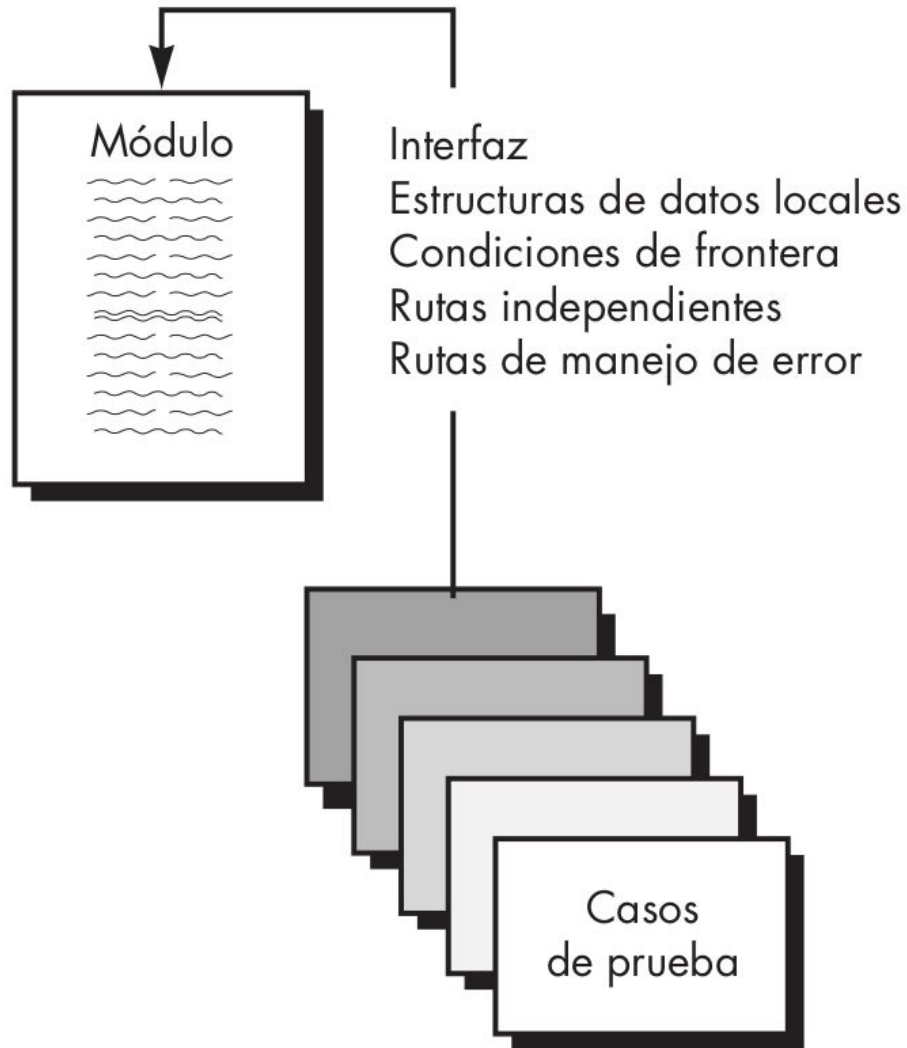
Alcance de la prueba



Prueba de Unidad

- Centrada en la verificación de la unidad más pequeña del diseño de software: el componente o módulo de software.
 - Ámbito restringido
- Se enfocan en la lógica de procesamiento interno y de las estructuras de datos que usa la unidad (**UUT**)
- Pueden realizarse en paralelo, por lo general es responsabilidad del programador.

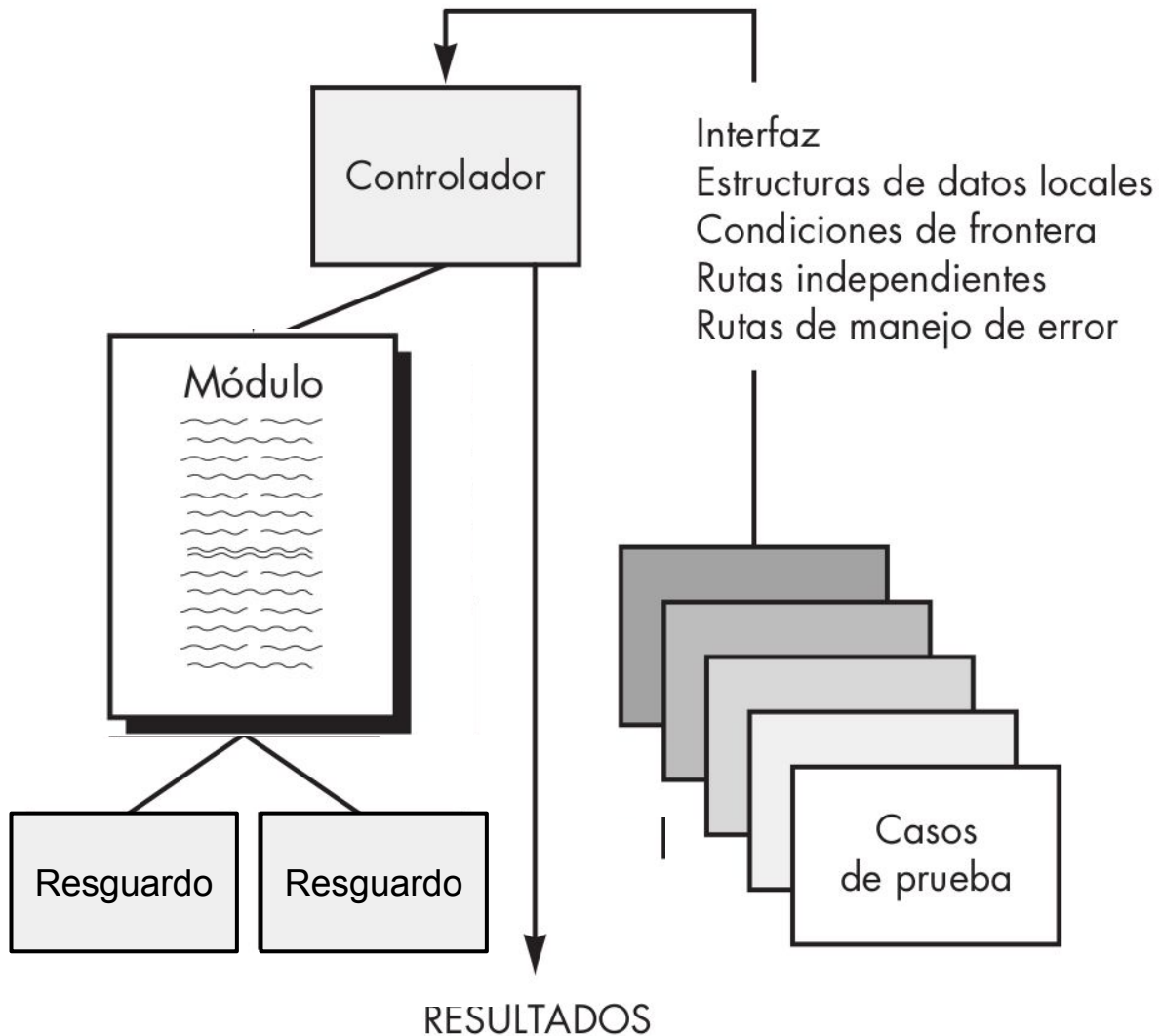
Prueba de Unidad: casos de prueba



Procedimiento de prueba de Unidad

- El diseño de las pruebas se puede hacer antes o después de la codificación de la unidad.
- Cada caso de prueba debe incluir un conjunto de resultados esperados
- Un componente/unidad no es un programa independiente
 - Unidades controladoras & unidades de resguardo
 - Añaden una “sobrecarga” a las pruebas (no forman parte del entregable).
 - Deben ser unidades simples de mantener!
- Las pruebas se simplifican cuando se prueba un componente con alta cohesión y bajo acoplamiento.

Prueba de Unidad: procedimiento



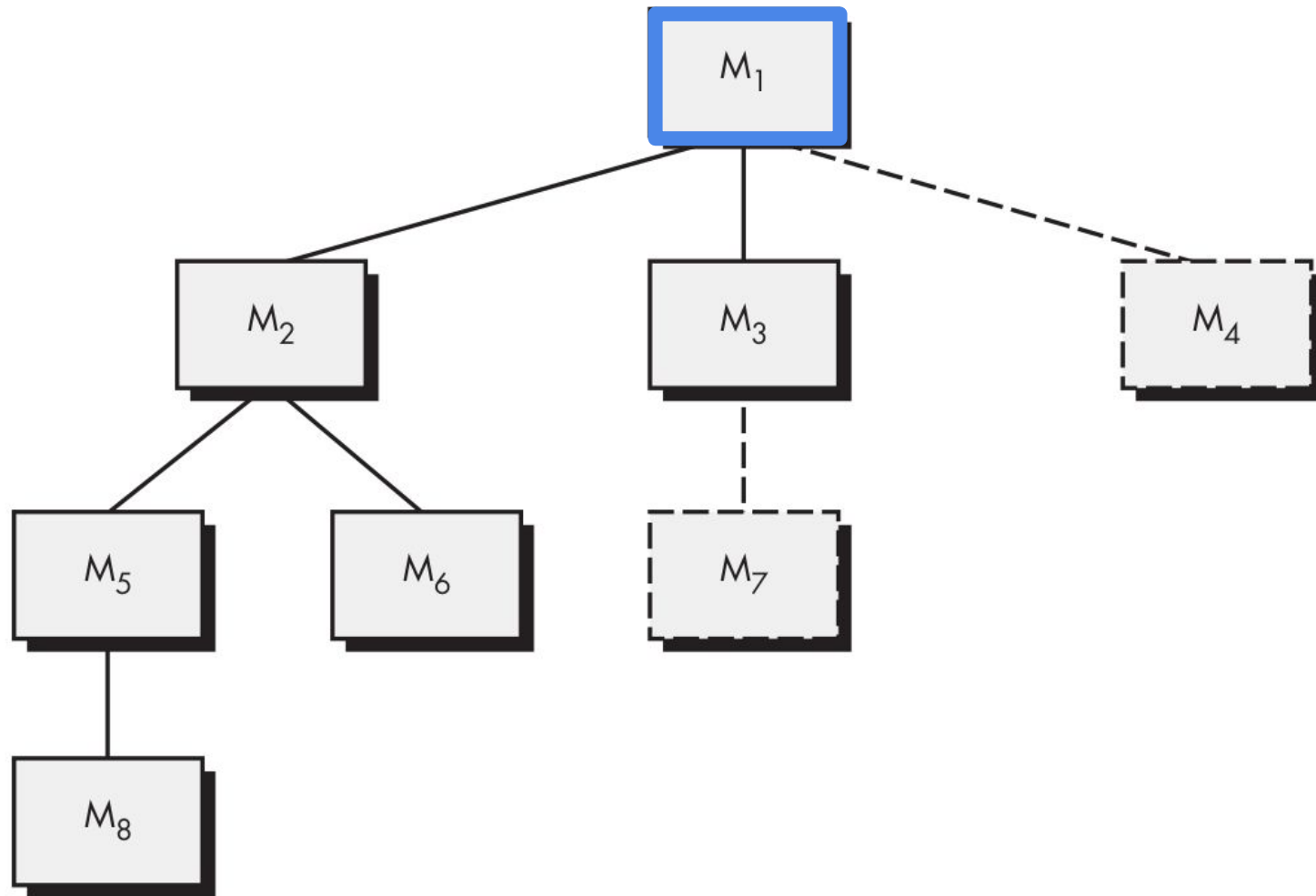
Prueba de Integración

- Construir el SW a través de una integración incremental
- Evitar una Integración Big Bang
- Facilita aislar y corregir errores de interfaz
- Posibilita una prueba más exhaustiva de interfaces
- Descubre errores asociados con la interfaz
- Enfoque incremental
 - Ascendente
 - Descendente
 - Mixto

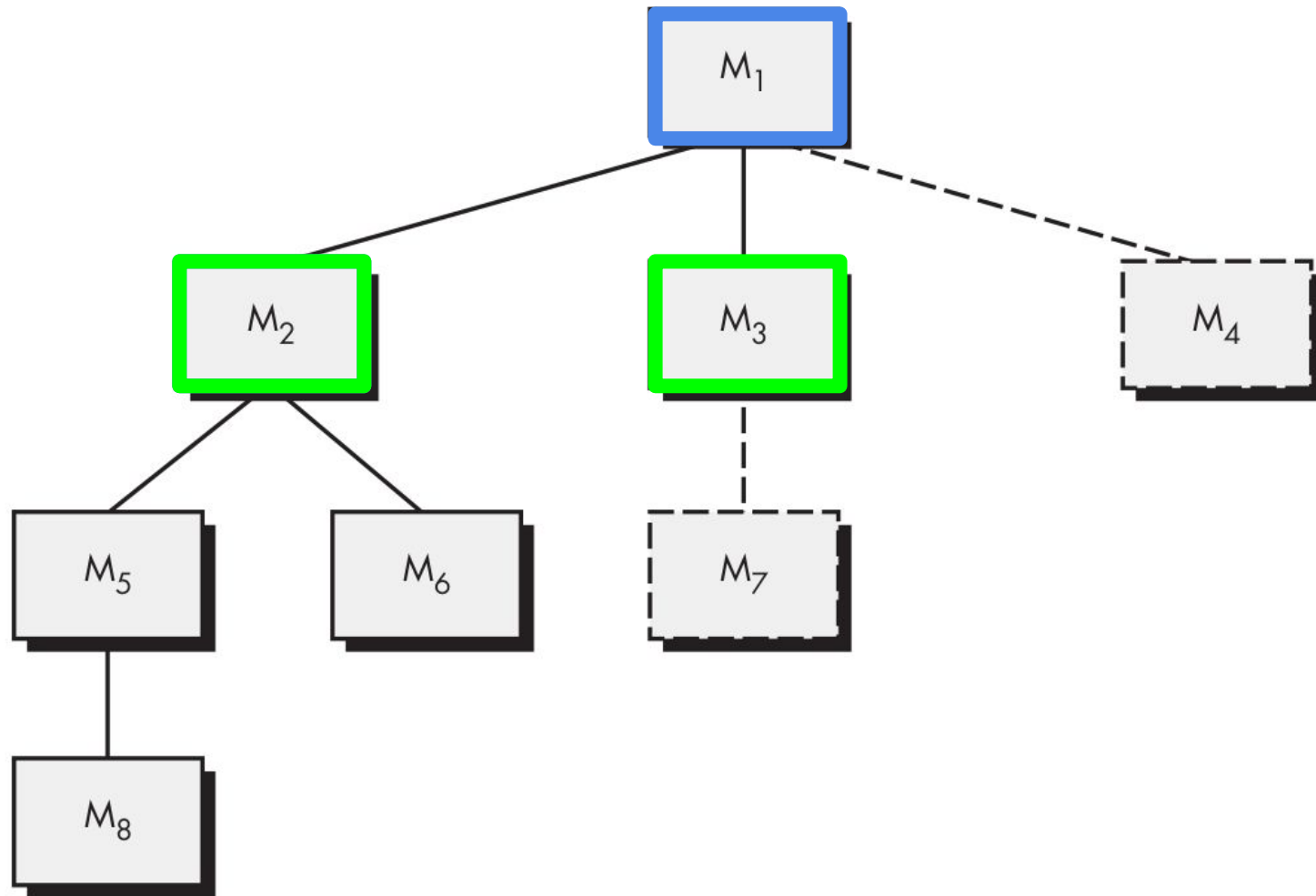
Prueba de Integración descendente

- Los módulos se integran al moverse hacia abajo a través de la jerarquía de control
- Se usa primero en profundidad o primero en anchura (Depth-first search or Breadth-first search)
- Verifica los principales puntos de control al principio
 - la toma de decisiones ocurre en niveles superiores en la jerarquía
- Pueden aparecer problemas logísticos: se requiere un proceso complejo subordinado para probar un módulo
 - Demorar las pruebas hasta que los resguardos se sustituyan con módulos reales (~Big Bang)
 - Desarrollar resguardos (puede ser complejo)
 - Integrar el software desde abajo hacia arriba.

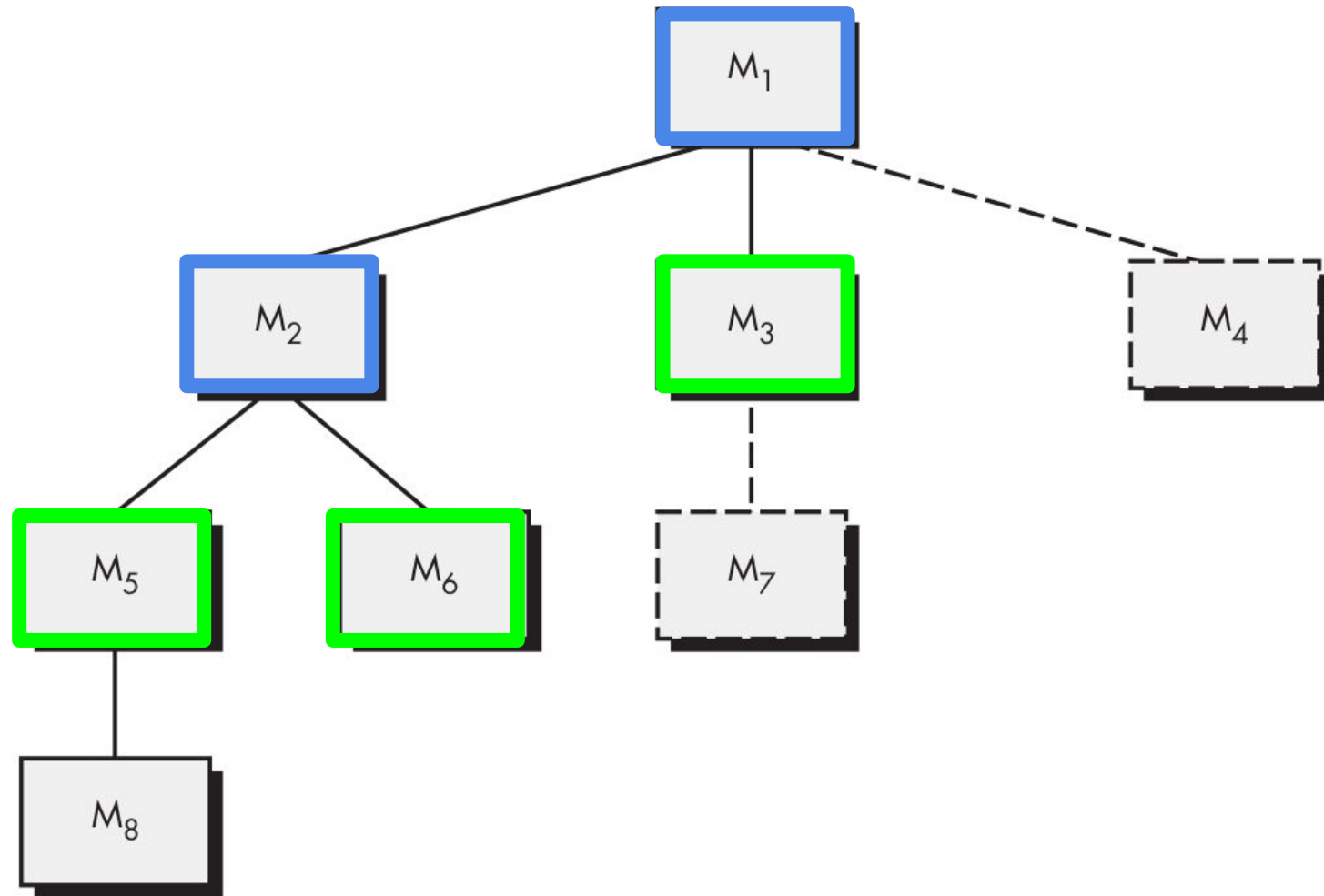
Prueba de Integración Incremental Descendente



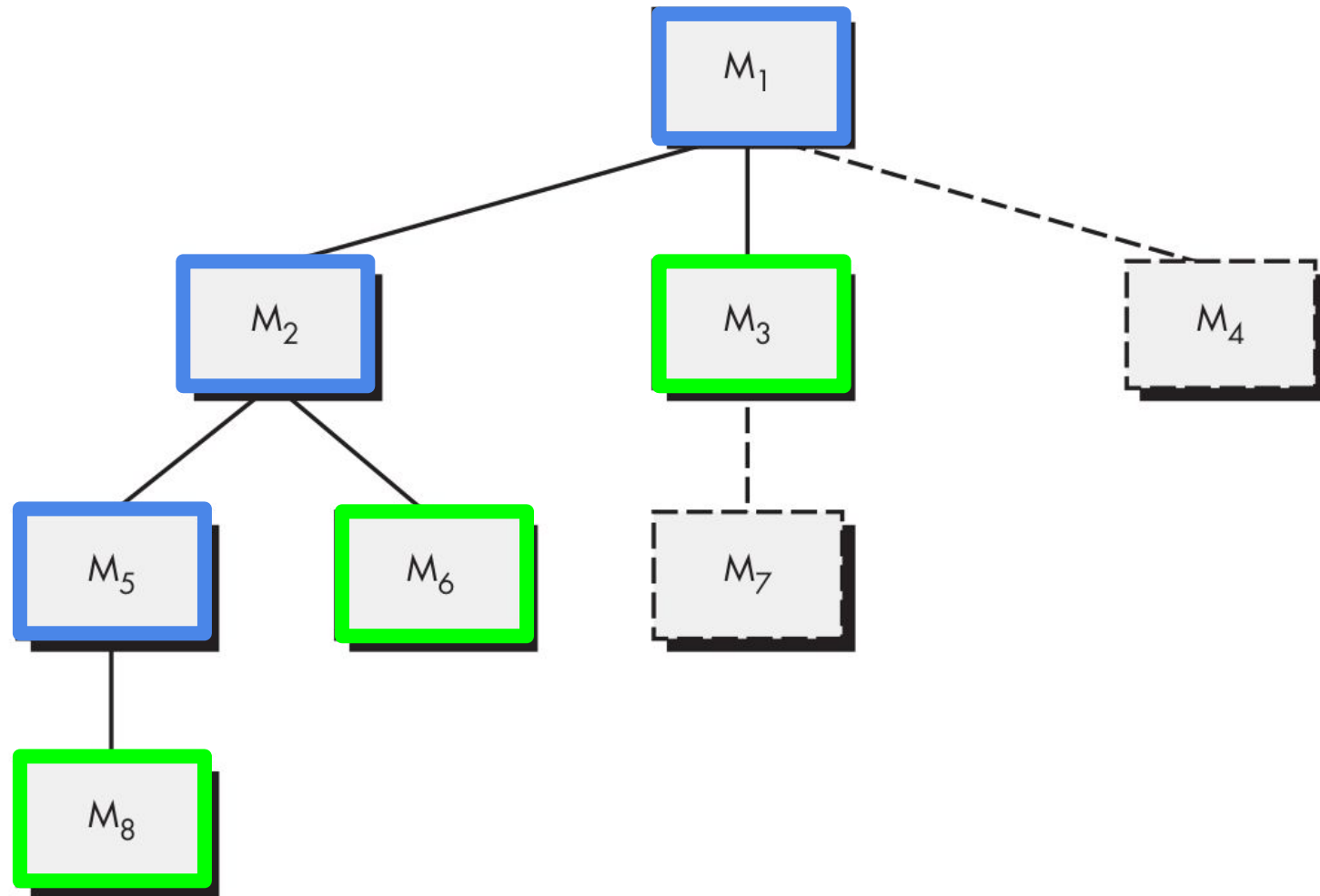
Prueba de Integración Incremental Descendente



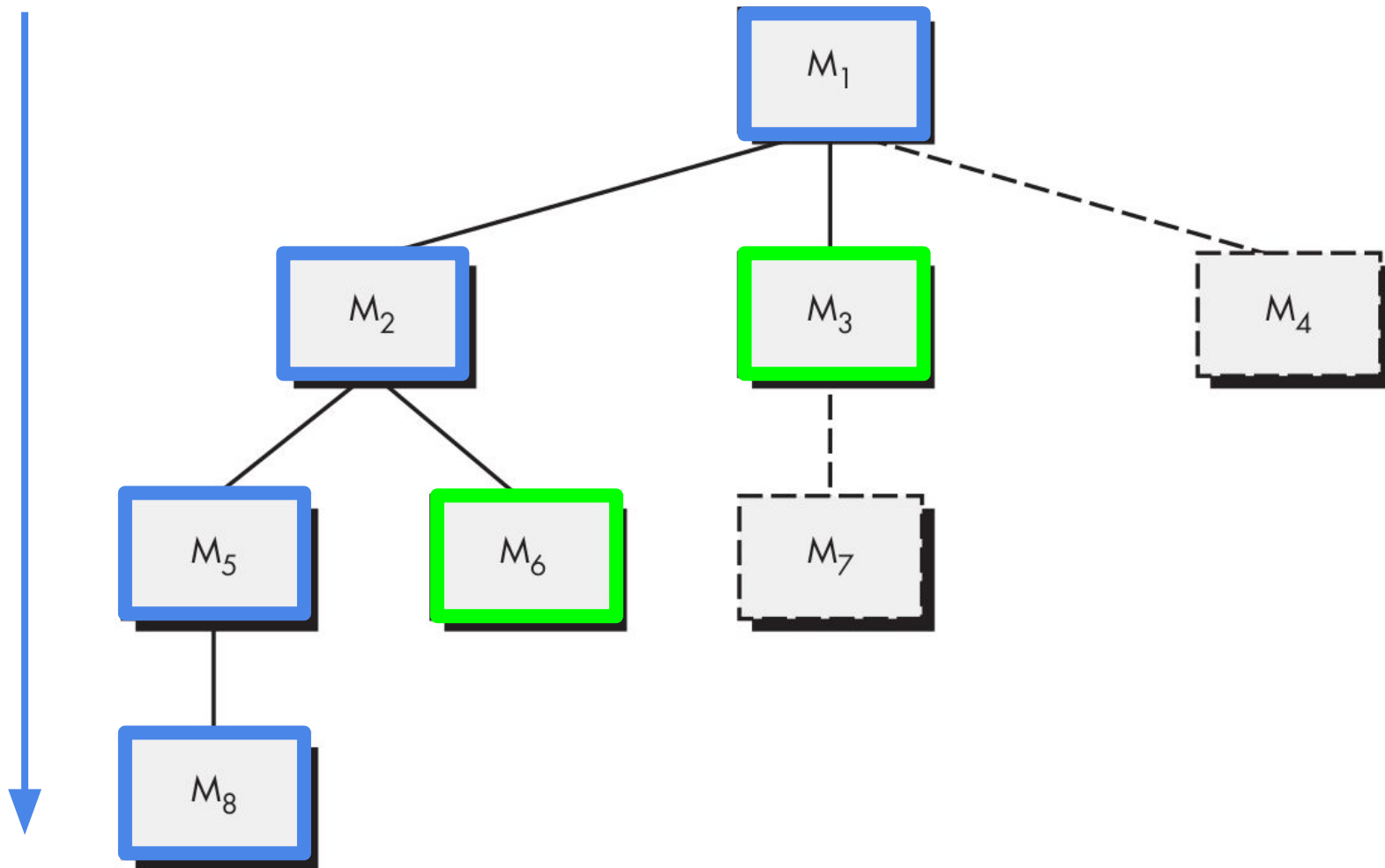
Prueba de Integración Incremental Descendente



Prueba de Integración Incremental Descendente



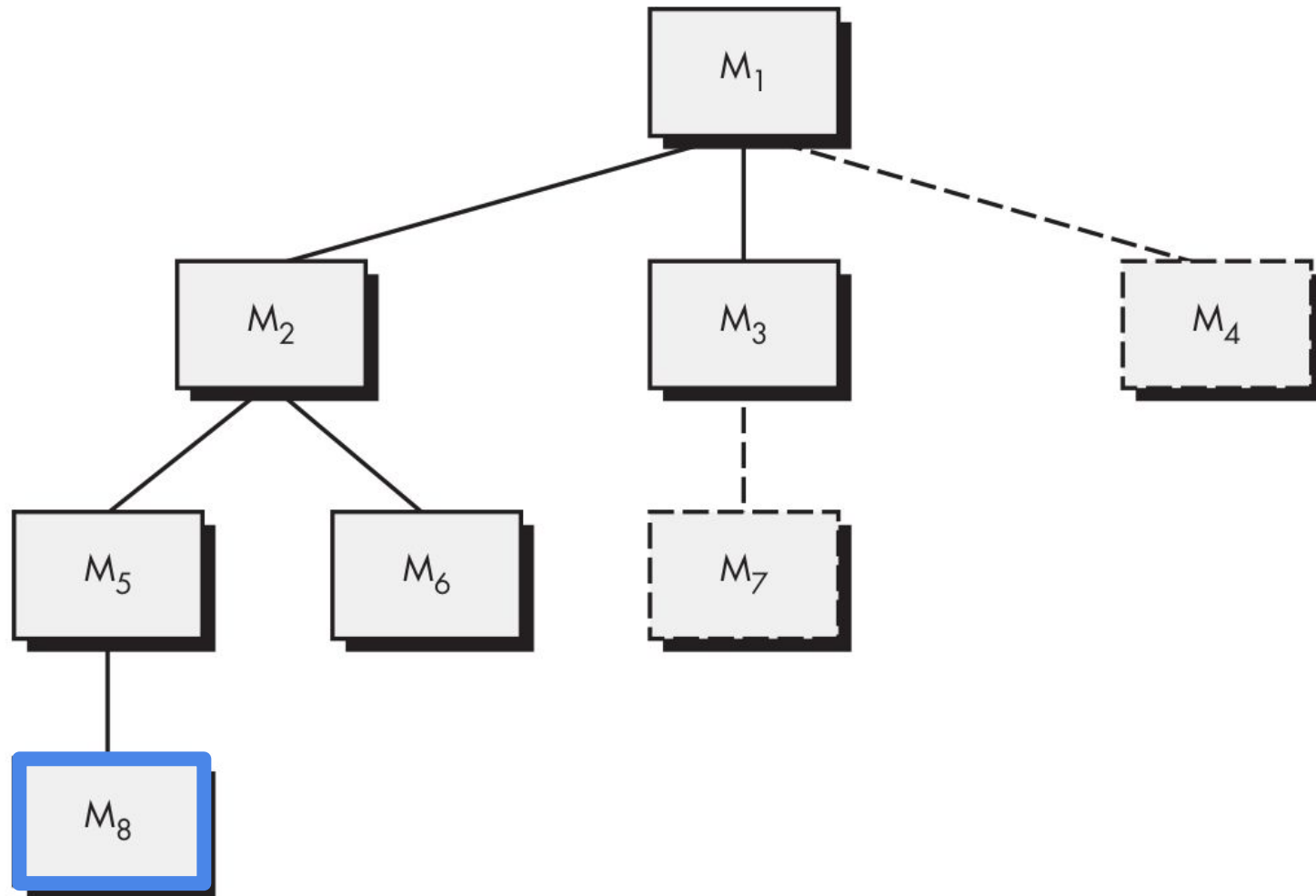
Prueba de Integración Incremental Descendente



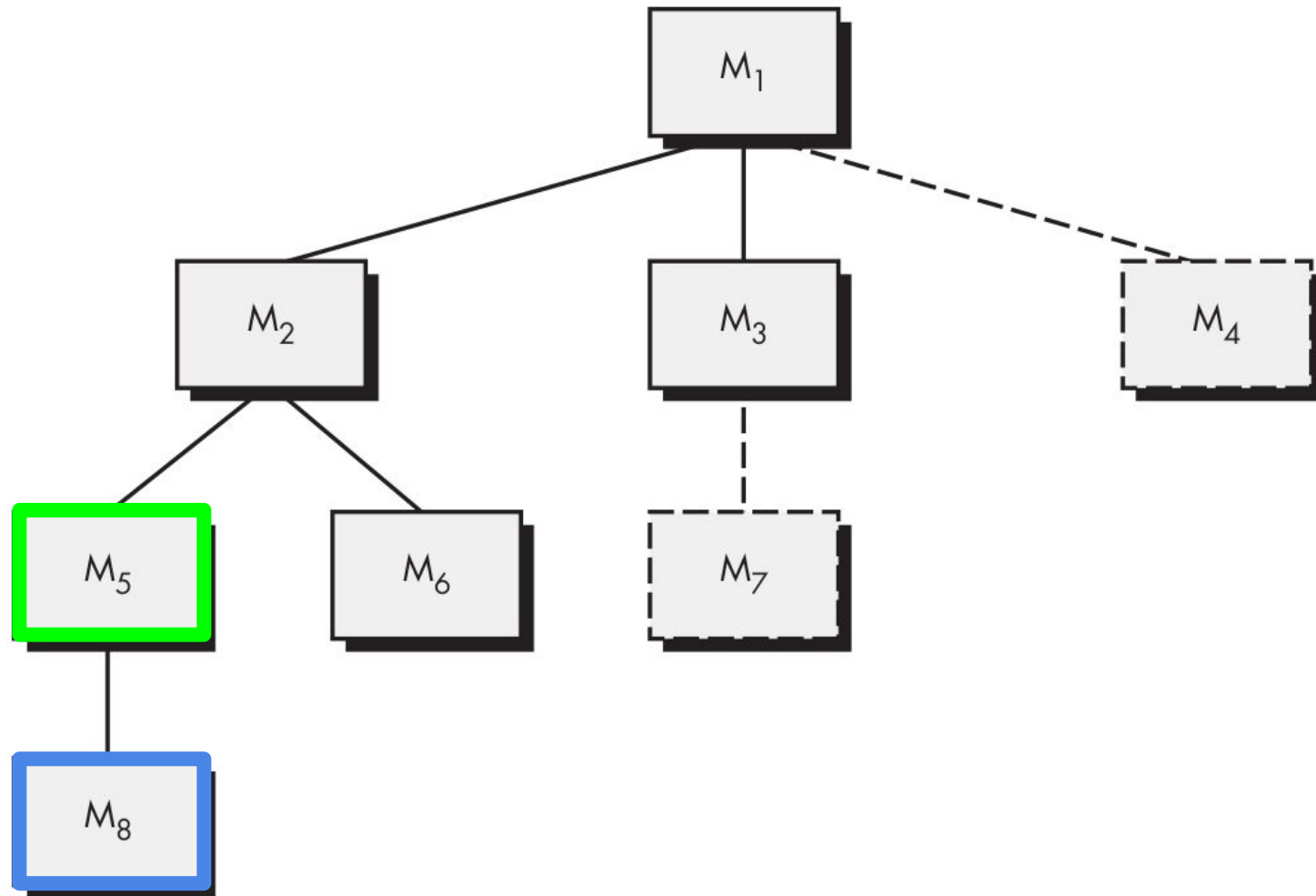
Prueba de Integración Incremental Ascendente

- Comienza con la prueba de módulos atómicos.
- La funcionalidad de los componentes más subordinados está disponible (puede haber grupos de módulos disponibles)
- Se elimina la necesidad de resguardos (stubs) pero se deben escribir los controladores (drivers)
- Parece natural desde el punto de vista del programador

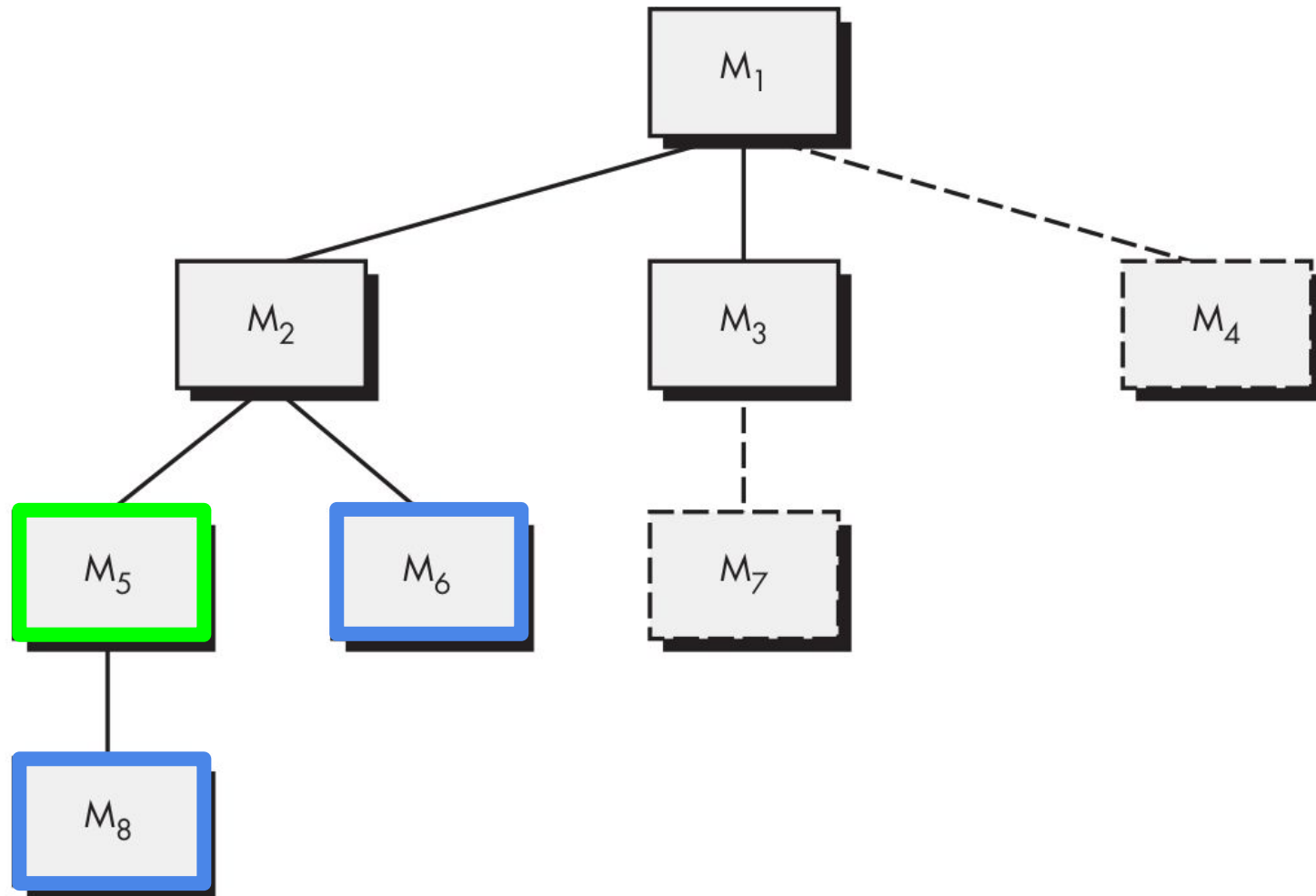
Prueba de Integración Incremental Ascendente



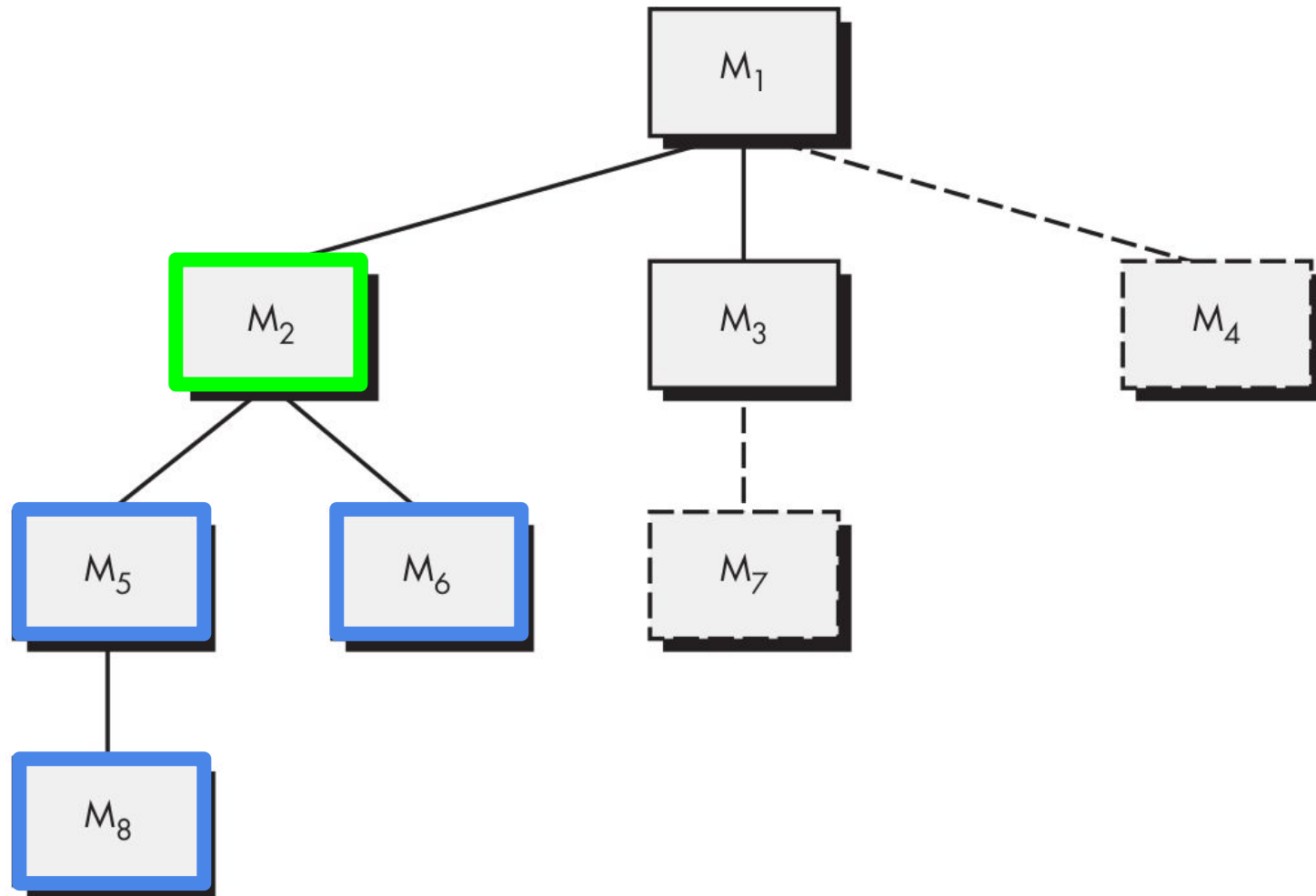
Prueba de Integración Incremental Ascendente



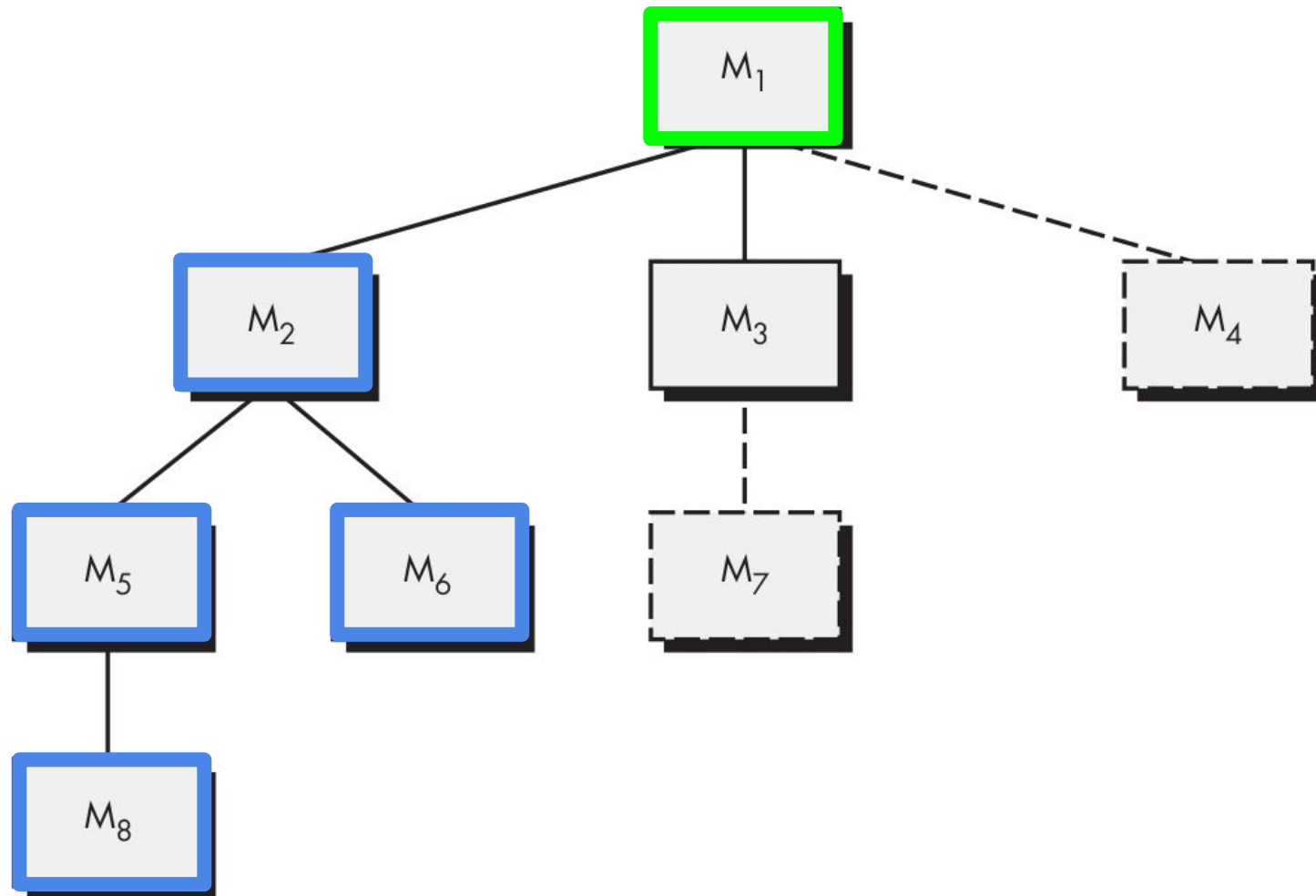
Prueba de Integración Incremental Ascendente



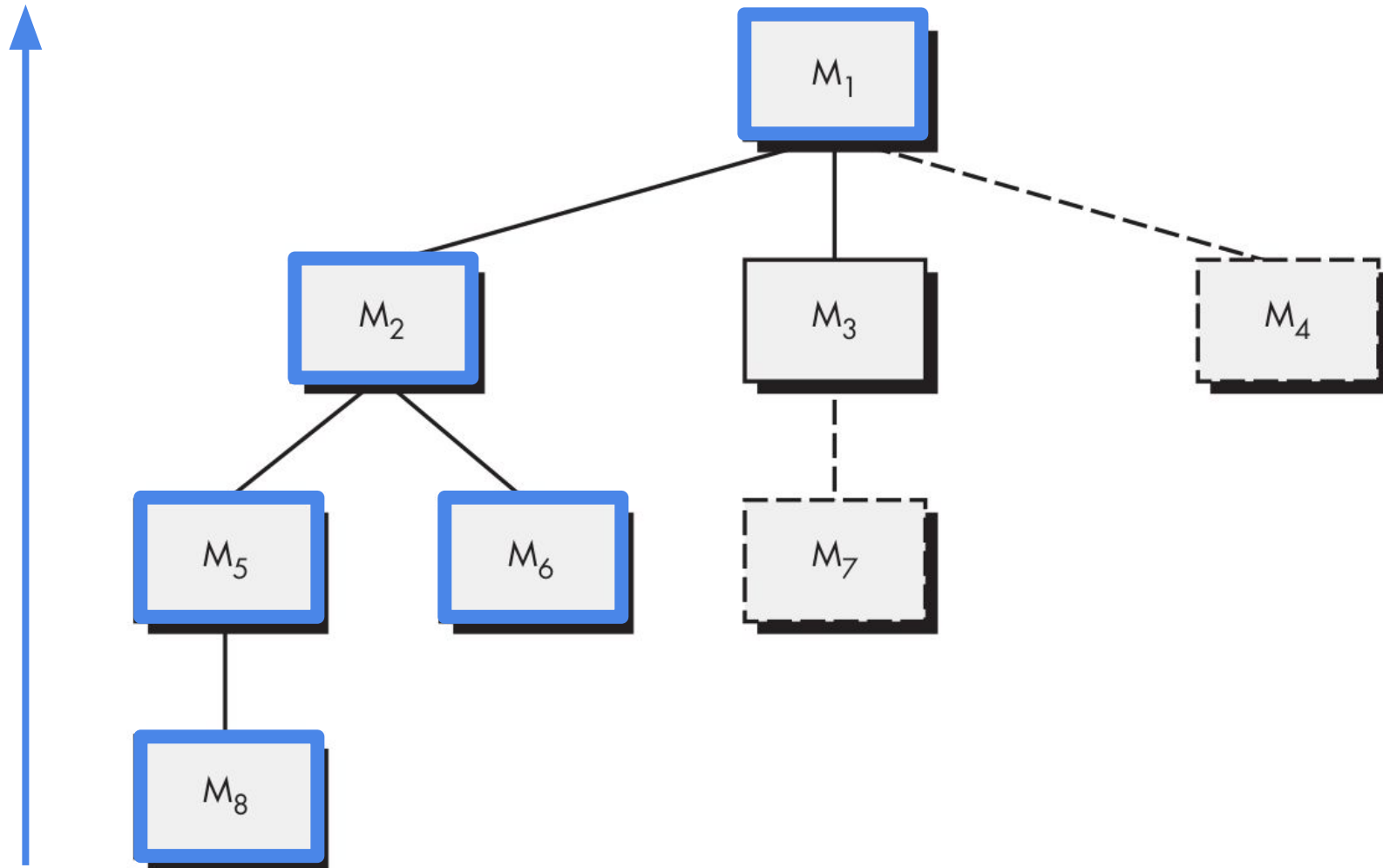
Prueba de Integración Incremental Ascendente



Prueba de Integración Incremental Ascendente



Prueba de Integración Incremental Ascendente



En una integración descendente, nombre una ventaja y desventaja de algunos de los enfoques incrementales de integración vistos.

¿Puede pensar un enfoque mixto?

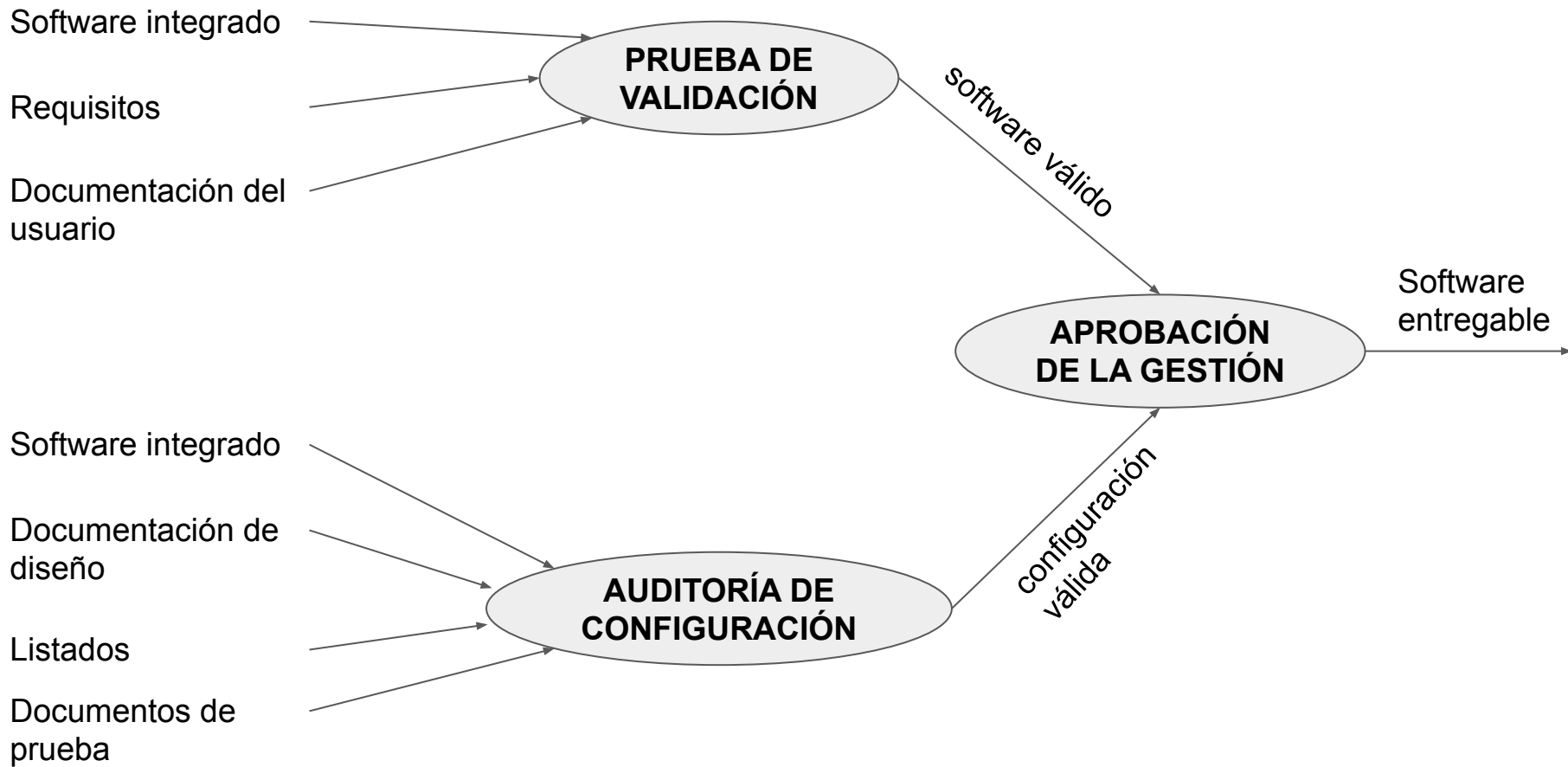
Prueba de regresión

- Es la ejecución repetida de pruebas que ya se realizaron
- Cada vez que se reemplaza un driver o stub, el SW cambia
 - Pueden introducirse problemas con las funciones anteriores
 - Necesidad de re-testear lo anterior
- Aseguran que los cambios no propagan efectos colaterales no deseados.
- Un cambio puede introducir comportamiento no planeado o errores adicionales
- Automatización de este tipo de pruebas es esencial

Prueba de validación

- El software se encuentra integrado
- Consiste en una prueba de caja negra a gran escala
- Su resultado demuestra la conformidad con los requisitos
 - Requisitos funcionales
 - Requisitos no funcionales (disponibilidad, seguridad, performance, mantenibilidad, etc)
 - Documentación correcta
- Los resultados pueden ser
 - Aceptables o no
 - Se genera una lista de deficiencias a corregir
- Estas pruebas exigen una auditoría de configuración

Entrega del software



Prueba Alfa y Beta

- Son consideradas **pruebas de aceptación**
- La **prueba Alfa** es conducida por el cliente en el lugar de desarrollo (ambiente controlado)
- La **prueba Beta** se lleva a cabo en ambientes del cliente y es conducida por el usuario final del sistema (entorno no controlado)

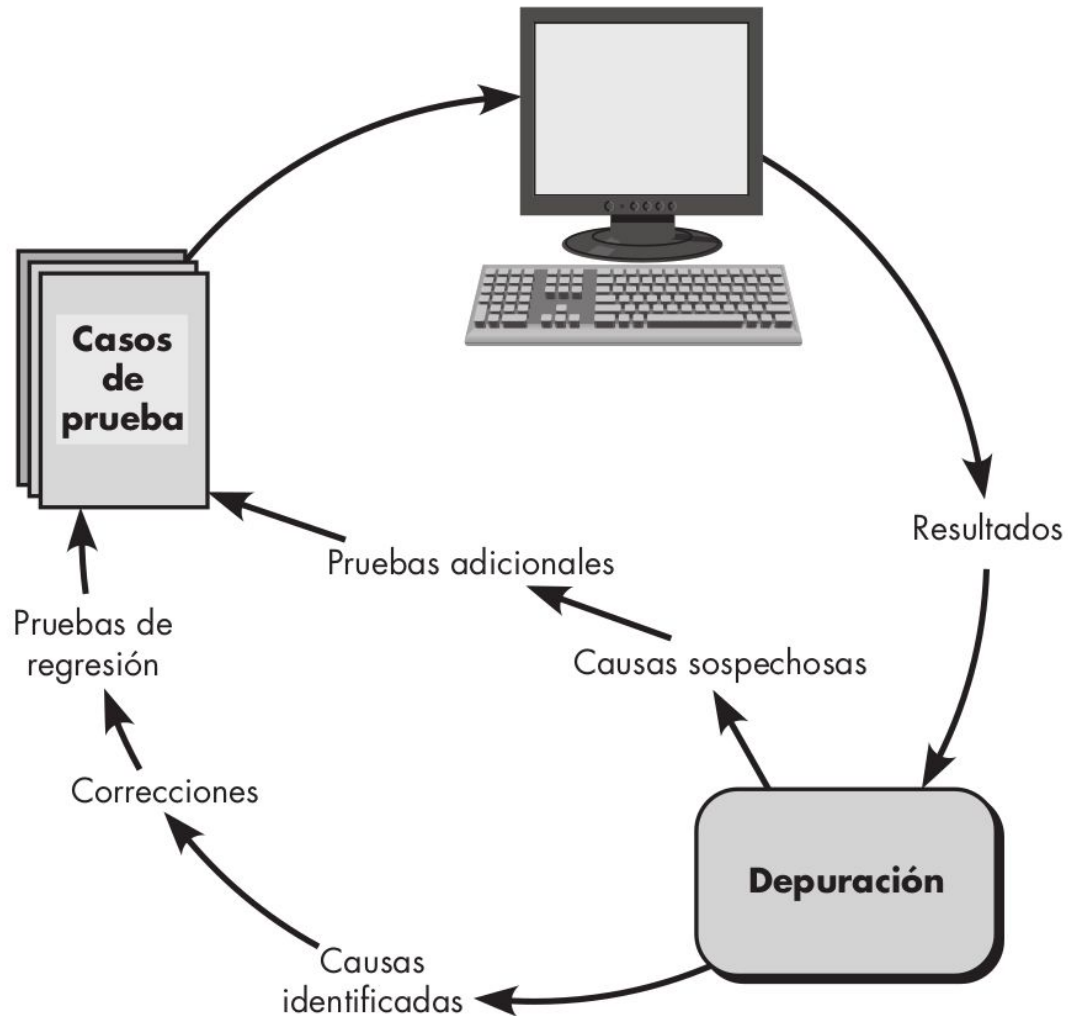
Pruebas del sistema

- Tiene como objetivo ejercitar profundamente al sistema
- El sistema de software que se construye es solo un elemento de un sistema más grande (hardware, servicios, gente, información, etc)
- Pruebas de recuperación
 - Recuperación ante fallas y reanudación (tiempo de rec.)
 - Tolerancia a fallas
 - Es automática en sistemas críticos
- Pruebas de seguridad
- Pruebas de estrés
- Pruebas de performance

El proceso de debugging

- La depuración no es una prueba, pero con frecuencia ocurre como consecuencia de una prueba
- Su objetivo es corregir la causa de los errores detectados
- En sistemas con componentes altamente acoplados puede ser un arte:
 - El síntoma y la causa pueden ser geográficamente remotos
 - El síntoma puede desaparecer (temporalmente) al corregir otro error
 - El síntoma en realidad puede no ser causado por errores
 - El síntoma puede ser causado por un error humano
 - El síntoma puede ser resultado de problemas de temporización más que de problemas de procesamiento.
 - Puede ser difícil reproducir con precisión las condiciones de entrada.
 - El síntoma puede ser intermitente.
 - El síntoma puede deberse a causas de paralelizar un proceso

El proceso de debugging



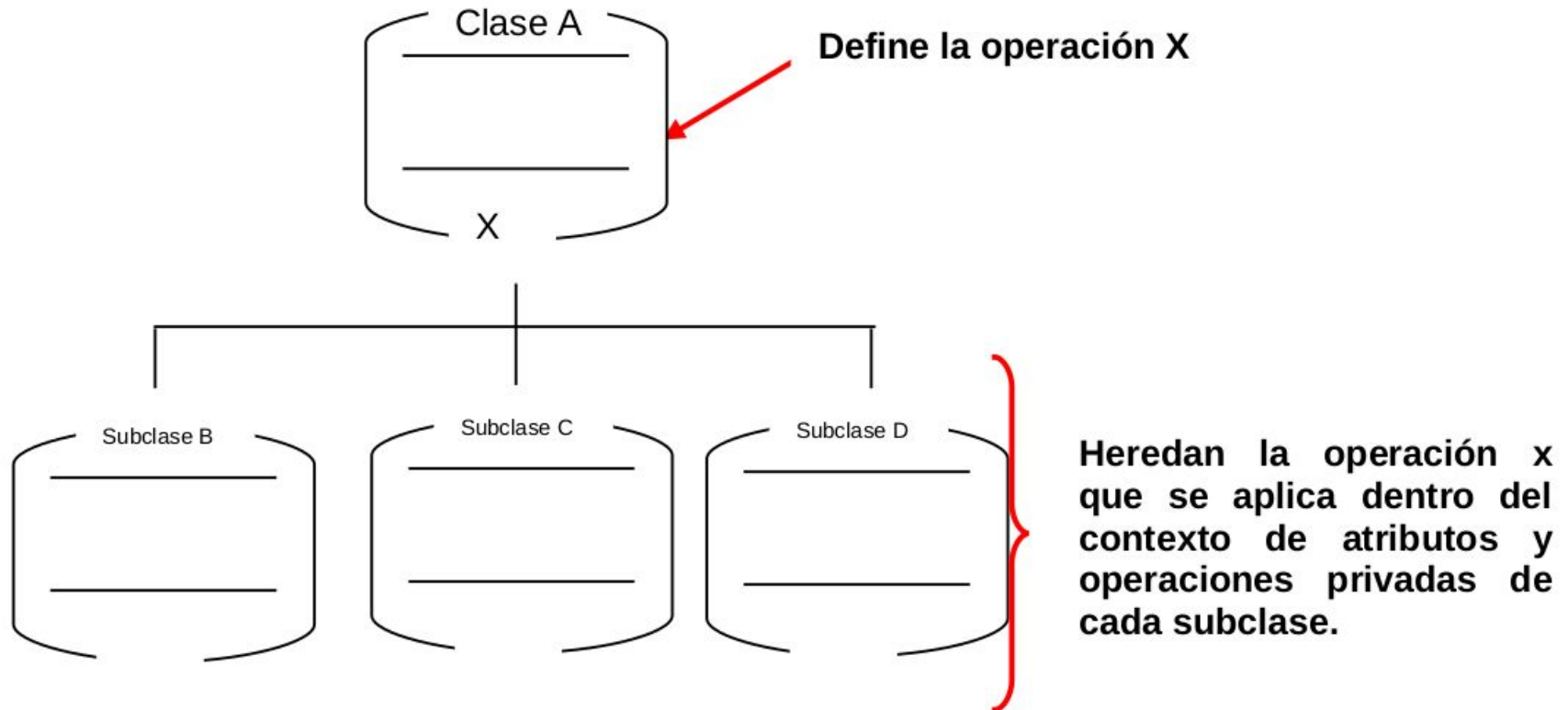
¿Qué tipos de pruebas nos pueden indicar con más precisión dónde está la causa del subyacente del error?

Estrategias de depuración (debugging)

- **Fuerza bruta**
 - Se agregan “salidas” en diferentes partes del programa.
 - Se ejecuta repetidamente hasta encontrar una pista.
- **Vuelta atrás**
 - Se comienza donde está el síntoma.
 - Se sigue el código hacia atrás hasta encontrar la falta.
 - Muchos caminos posibles, inmanejables algunas veces.
- **Eliminación de causas**
 - Por inducción o deducción
 - Se organizan los datos relacionados al error para aislar la causa
 - Se generan hipótesis de la causa y se usan los datos para rechazar o aceptar hipótesis (se crean nuevos casos de pruebas)

Pruebas para sistemas OO

- La unidad cambia: clases y objetos
 - Una clase es el foco de una prueba de unidad



Prueba para sistemas OO (pruebas unitarias)

- Una jerarquía de clases afecta las pruebas unitarias
 - Se prueba un método heredado en el contexto de cada contexto de subclase concreta
- Se pasa del detalle algorítmico de un módulo y los datos que fluyen, a la revisión de métodos visibles y estados posibles.

Prueba para sistemas OO (pruebas de integración)

- No hay una estructura jerárquica de control obvia
 - Casi no tiene sentido hablar de pruebas de integración descendentes y ascendentes
 - Se aplica tests de regresión para asegurar no haya efectos laterales (side effects)
- Pruebas basadas en hebras :
 - Integra el conjunto de clases requeridas para responder a una entrada o evento
 - Cada hilo o rodaja (program slice) es integrado y testeado individualmente
- Pruebas basadas en uso:
 - Se prueban primero las clases más independientes y después la capa de clases directamente dependientes
 - Continúa hasta construir todo el sistema.

Bibliografía

Pressman, Roger S. Software engineering: a practitioner's approach. 7th edition. Palgrave Macmillan, 2010.

Myers, Glenford J., Corey Sandler, and Tom Badgett. The art of software testing. John Wiley & Sons, 2011.