# Estrategias de prueba software

# Una mirada rápida

- ¿Qué es?
- ¿Quién lo hace?
- ¿Por qué es importante?
- ¿Cuáles son los pasos?
- ¿Cuál es el producto final?
- ¿Cómo me aseguro que lo hice bien?

# Un enfoque estratégico para la prueba de software

- Para realizar una prueba efectiva, debe realizar revisiones técnicas efectivas. Al hacerlo, eliminará muchos errores antes de comenzar la prueba.
- La prueba comienza en los componentes y opera “hacia afuera”, hacia la integración de todo el sistema de cómputo.
- Diferentes técnicas de prueba son adecuadas para distintos enfoques de ingeniería de software y en diferentes momentos en el tiempo.
- Las pruebas las realiza el desarrollador del software y (para proyectos grandes) un grupo de prueba independiente.
- Prueba y depuración son actividades diferentes, pero la depuración debe incluirse en cualquier estrategia de prueba.

# Verificación y validación

La prueba de software es un elemento de un tema más amplio que usualmente se conoce como verificación y validación (V&V).

La **verificación** se refiere al conjunto de tareas que garantizan que el software implementa correctamente una función específica.

La **validación** es un conjunto diferente de tareas que aseguran que el software que se construye sigue los requerimientos del cliente

# Organización de las pruebas de software

En todo proyecto de software hay un conflicto inherente de intereses que ocurre conforme comienzan las pruebas. Hoy en día, a las personas que construyen el software se les pide probarlo.

Desde un punto de vista psicológico, el análisis y diseño de software (junto con la codificación) son tareas constructivas. Las pruebas pueden considerarse como (psicológicamente) destructivas.

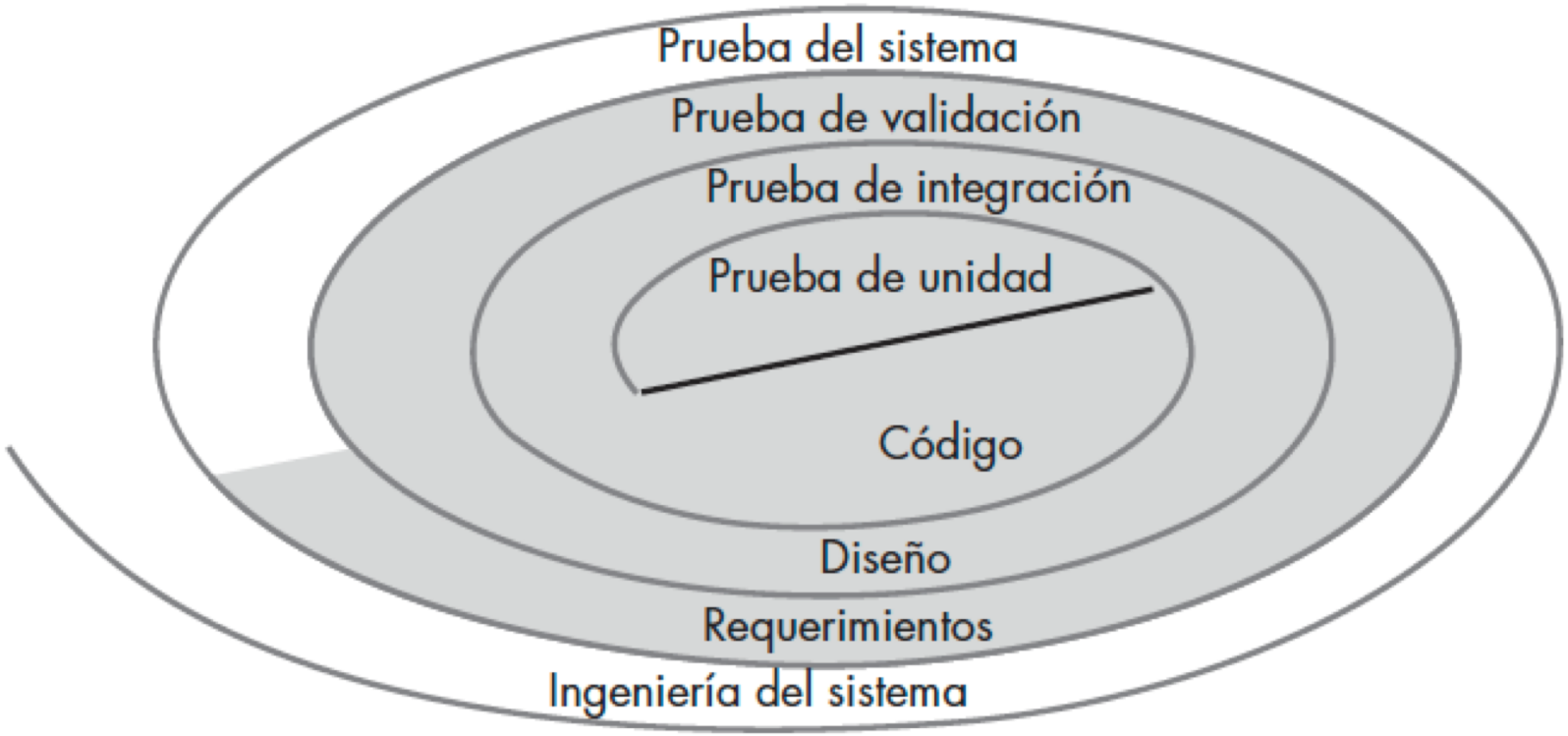
# Organización de pruebas software

El desarrollador de software siempre es responsable de probar las unidades individuales y de asegurarse de que cada una desempeña la función para la cual se diseñó.

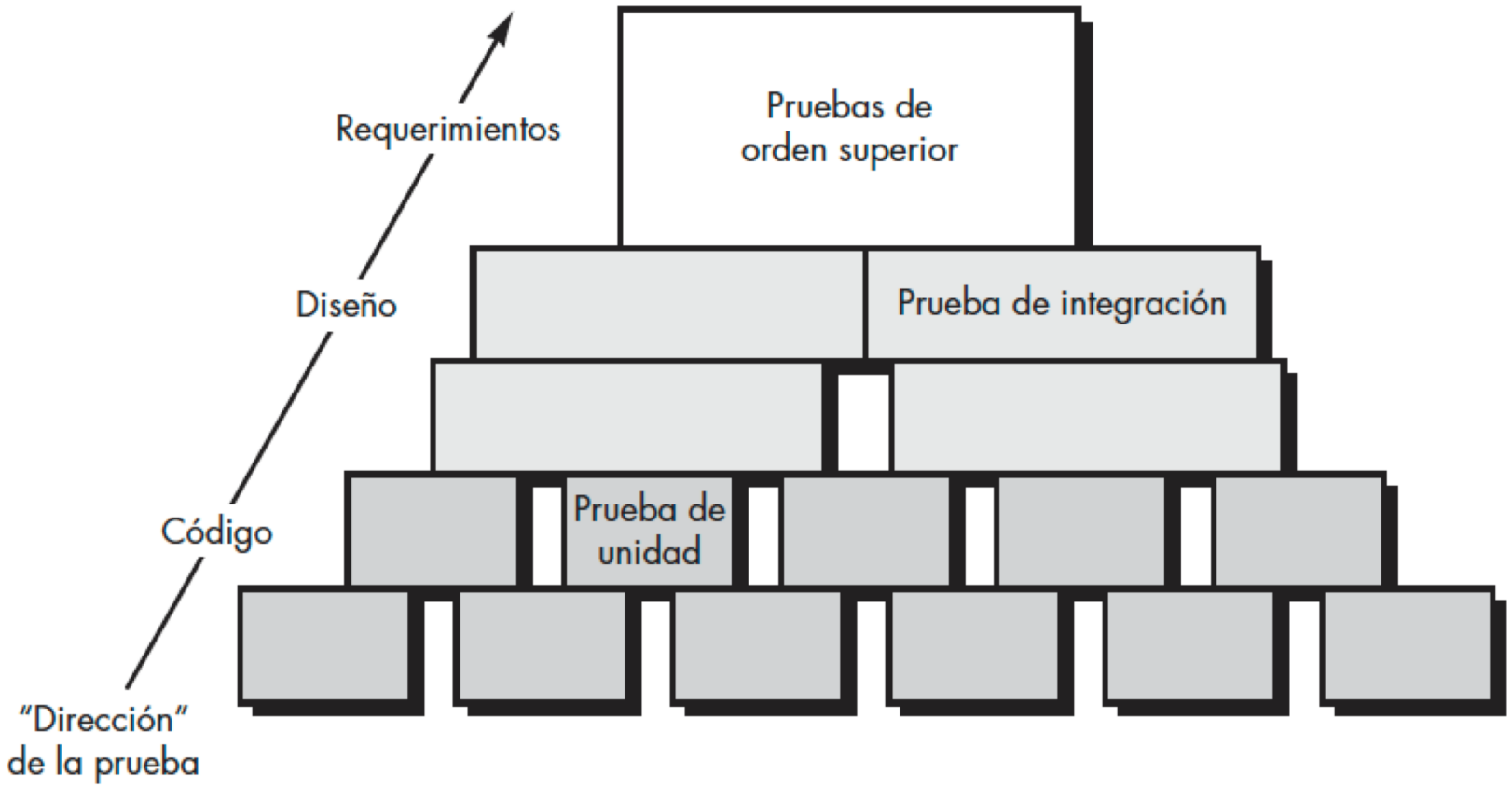
En muchos casos, el desarrollador también realiza pruebas de integración, una etapa en las pruebas que conduce a la construcción (y prueba) de la arquitectura completa del software.

Sólo después de que la arquitectura de software está completa se involucra un grupo de prueba independiente (GPI).

# Estrategia de prueba del software. Visión general



# Estrategia de prueba del software. Visión general



# Criterios para completar las pruebas

Pregunta clásica: “¿cuándo terminan las pruebas?, ¿cómo se sabe que se ha probado lo suficiente?”

No hay una respuesta definitiva a esta pregunta

Una respuesta a la pregunta es: “nunca se termina de probar; la carga pasa del ingeniero de software al usuario final”

# Aspectos estratégicos

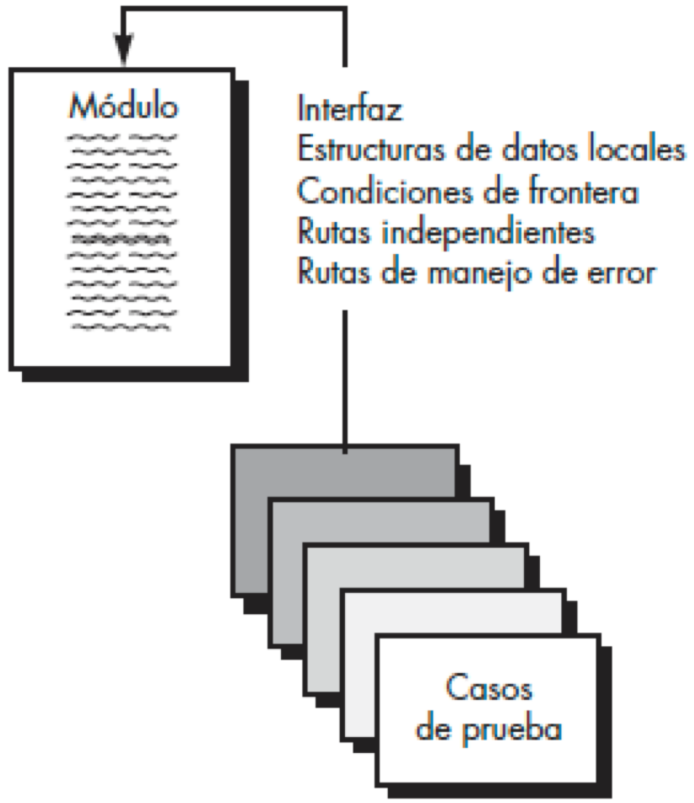
Tom Gilb arguye que una estrategia de software triunfará cuando quienes prueban el software:

- Especifican los requerimientos del producto en forma cuantificable mucho antes de comenzar con las pruebas
- Establecen de manera explícita los objetivos de las pruebas
- Entienden a los usuarios del software y desarrollan un perfil para cada categoría de usuario
- Desarrollan un plan de prueba que enfatice “pruebas de ciclo rápido”
- Construyen software “robusto” que esté diseñado para probarse a sí mismo
- Usan revisiones técnicas efectivas como filtro previo a las pruebas
- Realizan revisiones técnicas para valorar la estrategia de prueba y los casos de prueba

# Estrategias de prueba para software convencional

- Pruebas de unidad
- Pruebas de integración

# Pruebas de unidad



# Pruebas de unidad

Entre los potenciales errores que deben ponerse a prueba cuando se evalúa el manejo de errores están:

1) la descripción de error ininteligible,

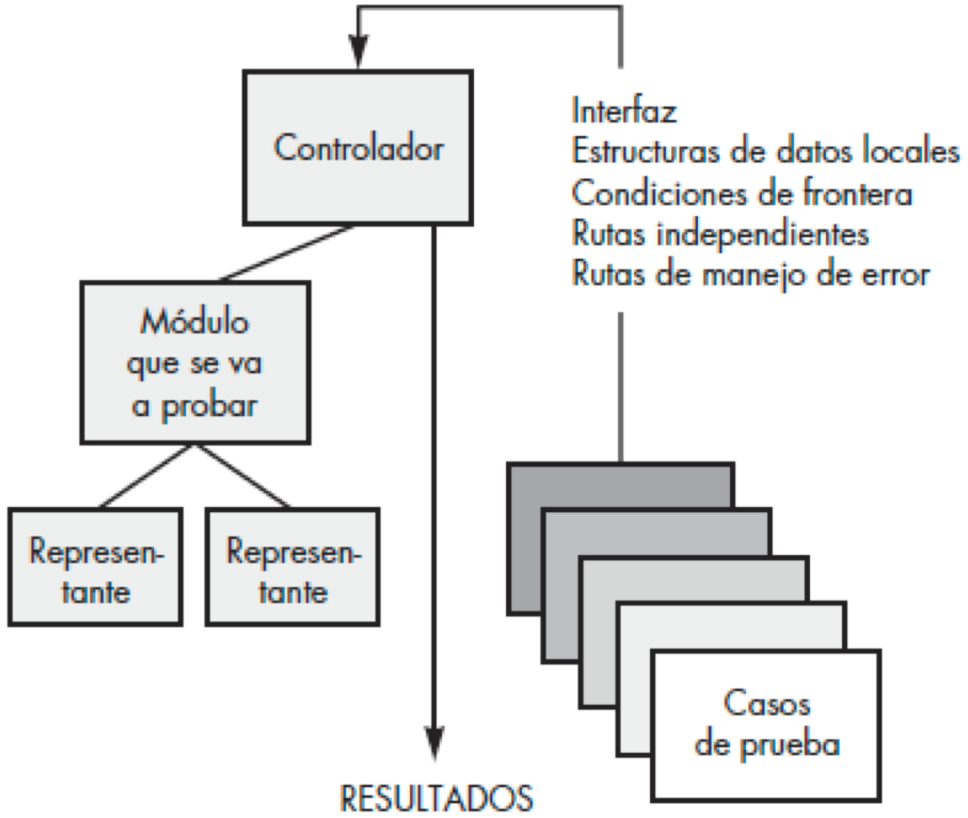
2) el error indicado no corresponde con el error que se encuentra,

3) la condición del error causa la intervención del sistema antes de manejar el error

4) el procesamiento excepción-condición es incorrecto y

5) la descripción del error no proporciona suficiente información para auxiliar en la localización de la causa del error.

# Pruebas de unidad

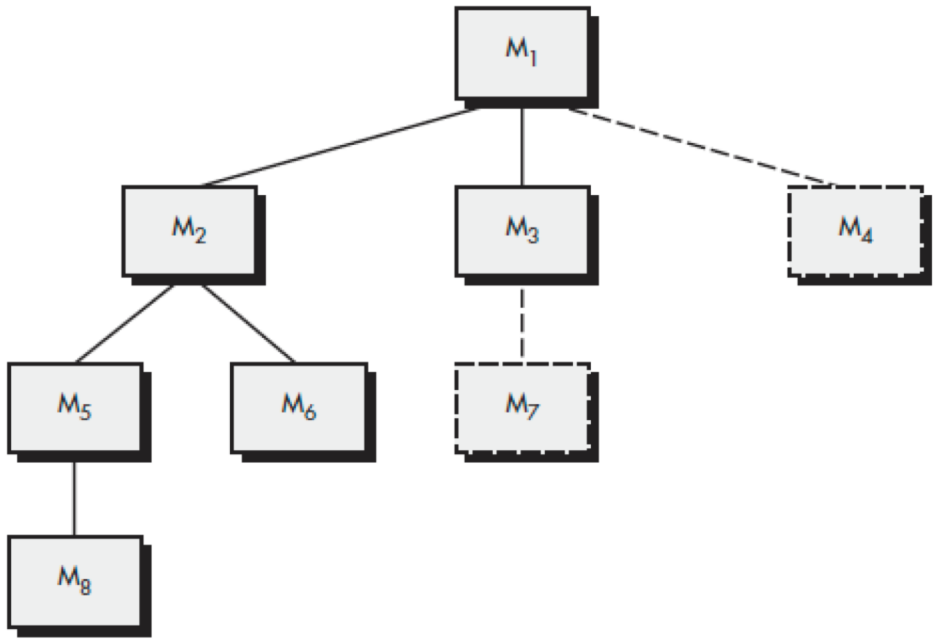


# Pruebas de integración

Si todos los módulos funcionan individualmente, ¿por qué dudan que funcionarán cuando se junten todos?

Las pruebas de integración son una técnica sistemática para construir la arquitectura del software mientras se llevan a cabo pruebas para descubrir errores asociados con la interfaz. El objetivo es tomar los componentes probados de manera individual y construir una estructura de programa que se haya dictado por diseño

# Pruebas de integración. Integración descendente




1. El módulo de control principal se usa como un controlador de prueba y los representantes (stubs) se sustituyen con todos los componentes directamente subordinados al módulo de control principal.

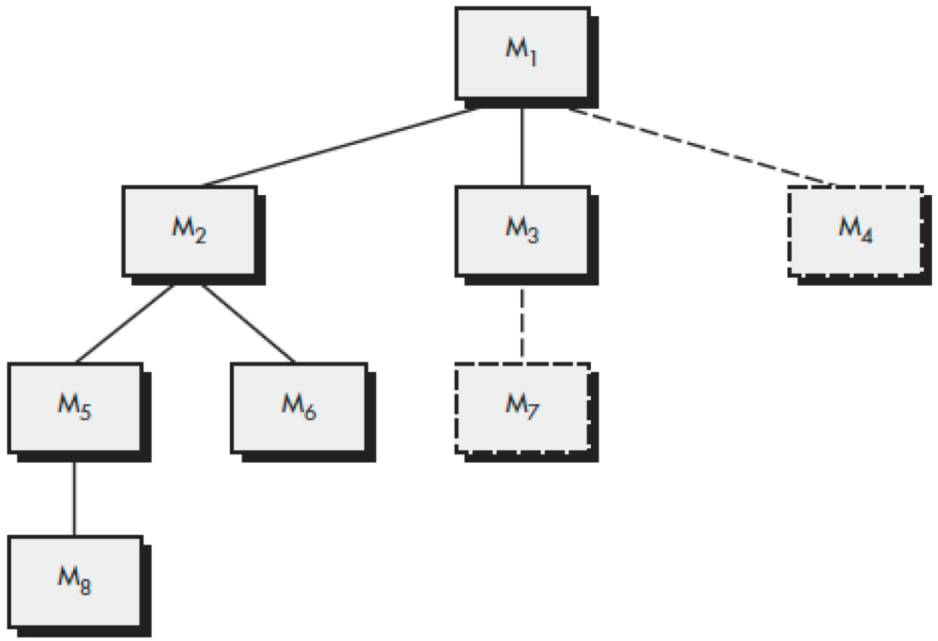
2. Dependiendo del enfoque de integración seleccionado (es decir, primero en profundidad o anchura), los representantes subordinados se sustituyen uno a la vez con componentes reales.

3. Las pruebas se llevan a cabo conforme se integra cada componente.

4. Al completar cada conjunto de pruebas, otro representante se sustituye con el componente real.

5. Las pruebas de regresión (que se analizan más adelante en este tema) pueden realizarse para asegurar que no se introdujeron nuevos errores

# Pruebas de integración. Integración descendente




1. El módulo de control principal se usa como un controlador de prueba y los representantes (stubs) se sustituyen con todos los componentes directamente subordinados al módulo de control principal.

2. Dependiendo del enfoque de integración seleccionado (es decir, primero en profundidad o anchura), los representantes subordinados se sustituyen uno a la vez con componentes reales.

3. Las pruebas se llevan a cabo conforme se integra cada componente.

4. Al completar cada conjunto de pruebas, otro representante se sustituye con el componente real.

5. Las pruebas de regresión (que se analizan más adelante en este tema) pueden realizarse para asegurar que no se introdujeron nuevos errores

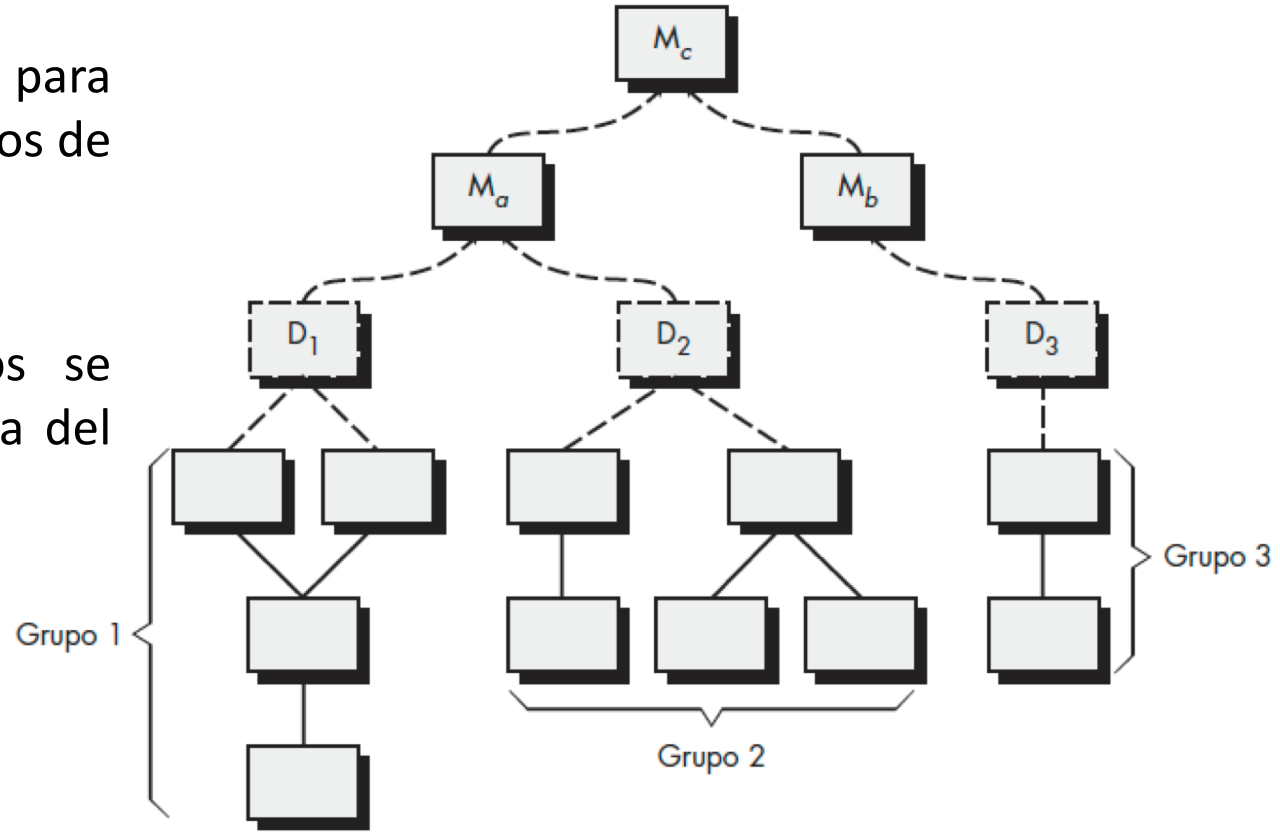
# Pruebas de integración. Integración ascendente

1. Los componentes en el nivel inferior se combinan en grupos (en ocasiones llamados construcciones o builds) que realizan una subfunción de software específica.

2. Se escribe un controlador (un programa de control para pruebas) a fin de coordinar la entrada y salida de casos de prueba.

3. Se prueba el grupo.

4. Los controladores se remueven y los grupos se combinan moviéndolos hacia arriba en la estructura del programa.



# Pruebas de integración. Prueba de regresión.

Cada vez que se agrega un nuevo módulo como parte de las pruebas de integración, el software cambia. Dichos cambios pueden causar problemas con las funciones que anteriormente trabajaban sin fallas.

La prueba de regresión es la nueva ejecución de algún subconjunto de pruebas que ya se realizaron a fin de asegurar que los cambios no propagaron efectos colaterales no deseados.

La suite de prueba de regresión contiene tres clases diferentes de casos de prueba:

- Una muestra representativa de pruebas que ejecutará las funciones de software.
- Pruebas adicionales que se enfocan en las funciones del software que probablemente resulten afectadas por el cambio.
- Pruebas que se enfocan en los componentes del software que cambiaron.

# Pruebas de integración. Prueba de humo.

Se diseña como un mecanismo de ritmo para proyectos críticos en el tiempo.

El enfoque de prueba de humo abarca las siguientes actividades:

1. Los componentes de software traducidos en código se integran en una construcción. Una construcción incluye todos los archivos de datos, bibliotecas, módulos reutilizables y componentes sometidos a ingeniería que se requieren para implementar una o más funciones del producto.

2. Se diseña una serie de pruebas para exponer los errores que evitarán a la construcción realizar adecuadamente su función. La intención debe ser descubrir errores “paralizantes” que tengan la mayor probabilidad de retrasar el proyecto.

3. La construcción se integra con otras construcciones, y todo el producto se somete a prueba de humo diariamente. El enfoque de integración puede ser descendente o ascendente.

# Pruebas de integración. Prueba de humo.

La prueba de humo proporciona algunos beneficios cuando se aplica sobre proyectos de software complejos y cruciales en el tiempo:

- Se minimiza el riesgo de integración.
- La calidad del producto final mejora.
- El diagnóstico y la corrección de errores se simplifican.
- El progreso es más fácil de valorar.

# Pruebas de integración. Opciones estratégicas

Conforme se realiza la integración, quien efectúa la prueba debe identificar los módulos críticos. Un módulo crítico tiene una o más de las siguientes características:

- aborda muchos requerimientos de software,
- tiene un alto nivel de control
- es complejo o proclive al error o
- tiene requerimientos de rendimiento definidos. Los módulos críticos deben probarse tan pronto como sea posible.

Además, las pruebas de regresión deben enfocarse en la función del módulo crítico.

# Estrategias de prueba para software orientado a objetos

- Prueba de unidad en el contexto Orientado a Objetos
- Prueba de integración en el contexto Orientado a Objetos

# Prueba de unidad en el contexto Orientado a Objetos

- Por lo general, una clase encapsulada es el foco de la prueba de unidad. Sin embargo, las operaciones dentro de la clase son las unidades comprobables más pequeñas.
- No es posible probar una sola operación en aislamiento  si más bien como parte de una clase.
- La prueba de clase para software OO es el equivalente de la prueba de unidad para software convencional.

# Prueba de integración en el contexto Orientado a Objetos

- La prueba basada en hebra

  integra el conjunto de clases requeridas para responder a una entrada o evento para el sistema

- La prueba basada en uso

  comienza la construcción del sistema al probar clases independientes para después pasar a usar las clases independientes

# Estrategias de prueba para aplicaciones web

1. El modelo de contenido para la aplicación web se revisa para descubrir errores.
2. El modelo de interfaz se revisa para garantizar que todos los casos de uso pueden adecuarse.
3. El modelo de diseño para la aplicación web se revisa para descubrir errores de navegación.
4. La interfaz de usuario se prueba para descubrir errores en los mecanismos de presentación y/o navegación.
5. A cada componente funcional se le aplica una prueba de unidad.

# Estrategias de prueba para aplicaciones web

6. Se prueba la navegación a lo largo de toda la arquitectura.
7. La aplicación web se implementa en varias configuraciones ambientales diferentes y se prueba en su compatibilidad con cada configuración.
8. Las pruebas de seguridad se realizan con la intención de explotar vulnerabilidades en la aplicación web o dentro de su ambiente.
9. Se realizan pruebas de rendimiento.
10. La aplicación web se prueba mediante una población de usuarios finales controlada y monitoreada. Los resultados de su interacción con el sistema se evalúan por errores de contenido y navegación, preocupaciones de facilidad de uso, preocupaciones de compatibilidad, así como confiabilidad y rendimiento de la aplicación web.

# Pruebas de validación

- Criterios de pruebas
- Pruebas alfa y beta

# Criterios de pruebas

La validación del software se logra con una serie de pruebas que demuestran conformidad con los requerimientos

Un plan de prueba tiene que garantizar que:

- Se satisfacen todos los requerimientos de funcionamiento
- Todo el contenido es preciso y se presenta de manera adecuada

# Pruebas de validación. Pruebas alfa y beta

**Prueba alfa**: se lleva a cabo en el sitio del desarrollador por un grupo representativo de usuarios finales

**Prueba beta**: se realiza en uno o más sitios del usuario final. Es una aplicación “en vivo” del software en un ambiente que no puede controlar el desarrollador

# Pruebas del sistema

- Pruebas de recuperación
- Pruebas de seguridad
- Pruebas de esfuerzo
- Pruebas de rendimiento
- Pruebas de despliegue

# Pruebas de recuperación

Los sistemas deben recuperarse de fallos y reanudar el procesamiento con poco o ningún tiempo de inactividad.

En ocasiones un sistema debe ser tolerante a las fallos, es decir, las fallos del procesamiento no deben causar el cese del funcionamiento del sistema global.

La prueba de recuperación es una prueba del sistema que fuerza al software a fallar en varias formas y que verifica que la recuperación se realice de manera adecuada

# Pruebas de seguridad

Cualquier sistema basado en software que gestione información sensible o cause acciones que puedan dañar de manera inadecuada a individuos es un blanco de penetración.

Durante la prueba de seguridad, quien realiza la prueba juega el papel del individuo que desea penetrar al sistema.

El papel del diseñador de sistemas es hacer que el costo de la penetración sea mayor que el valor de la información que se obtendrá.

# Pruebas de esfuerzo

Las pruebas de esfuerzo se diseñan para enfrentar los programas con situaciones anormales. Quien realiza las pruebas de esfuerzo pregunta: “¿cuánto podemos doblar esto antes de que se rompa?”.

Una variación de la prueba de esfuerzo es una técnica llamada prueba de sensibilidad. Esta intenta descubrir combinaciones de datos dentro de clases de entrada válidas que puedan causar inestabilidad o procesamiento inadecuado.

# Pruebas de rendimiento

Principalmente pensadas para sistemas en tiempo real y sistemas embebidos.

Es necesario que todos los elementos del sistema están plenamente integrados cuando puede determinarse el verdadero rendimiento de un sistema.

Suelen usarse junto con pruebas de esfuerzo y se monitoriza tanto el hardware como el software.

# Pruebas de despliegue

En muchos casos el software debe ejecutarse en diferentes plataformas y en sistemas operativos diferentes.

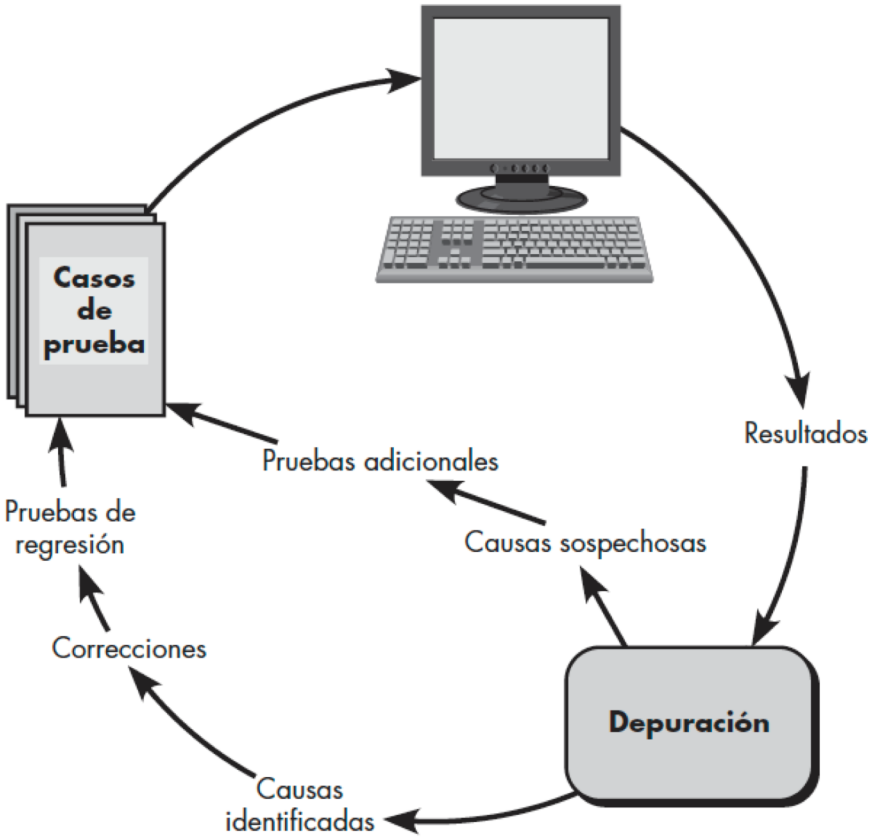
La prueba de despliegue ejecuta el software en cada entorno en el que se ejecutará.

# El arte de la depuración

La depuración ocurre como consecuencia de las pruebas “exitosas”. Es decir, cuando un caso de prueba descubre un error.

La depuración es el proceso que se da como resultado de la  corrección del error.

# El proceso de depuración



# Consideraciones psicológicas

*“La depuración es una de las partes más frustrantes de la programación. Tiene elementos de resolución de problemas o rompecabezas, junto con el desconcertante reconocimiento de que se cometió un error. La elevada ansiedad y la falta de voluntad para aceptar la posibilidad de los errores aumentan la dificultad de la tarea. Por fortuna, hay un gran alivio y la tensión se aligera cuando finalmente el error… se corrige.”*

Schneiderman

# Estrategias de depuración

*"La depuración es una aplicación directa del método científico que se ha desarrollado durante más de 2 500 años. La base de la depuración es localizar la fuente del problema [la causa] mediante una partición binaria, a través del trabajo con hipótesis que predicen nuevos valores por examinar."* Bradley

En general existen tres estrategias de depuración:

- Fuerza bruta
- Vuelta atrás
- Eliminación de causas

# Estrategias de depuración. Tácticas

- **Fuerza bruta**: se toman copias de la memoria, se invocan rastreros en el tiempo de ejecución…. La idea es recopilar toda la información posible para que en algún lugar de toda esa información se encuentre una pista que conduzca a la causa de un error.
- **El seguimiento hacia atrás**: este concepto consiste en ir al lugar donde se descubrió el error y se rastrea hacia atrás hasta que se encuentra la causa.
- **La eliminación de la causa**: se introduce el concepto de de partición binaria. Los datos relacionados con la ocurrencia del error se organizan para aislar sus causas.

# Estrategias de depuración. Automatización

*“Se han propuesto muchos nuevos enfoques y están disponibles muchos entornos de depuración comerciales. Los entornos de desarrollo integrados (IDE) brindan una forma de capturar algunos de los errores predeterminados específicos del lenguaje (por ejemplo, falta de caracteres de fin de sentencia , variables indefinidas, etc.) sin requerir compilación”* Hailpern y Santhanam

# Corrección de error

Antes de corregir un error deben plantearse una serie de cuestiones:

¿La causa del error se reproduce en otra parte del programa?

¿Qué “siguiente error” puede introducirse con la corrección que está a punto de realizar?

¿Qué debió hacerse para evitar este error desde el principio?

# Resumen

- Las pruebas de software representan el porcentaje más grande de esfuerzo técnico en el proceso de software.
- El objetivo de las pruebas del software es descubrir errores.
- La estrategia para probar software orientado a objetos comienza con pruebas que ejecutan las operaciones dentro de una clase y,  a partir de ahí se avanza hacia las de integración.
- Las aplicaciones web se prueban en forma muy parecida a los sistemas OO.
- Al comenzar con una indicación de un problema, la actividad de depuración debe rastrear la causa de un error.