UTN.BA
FACULTAD REGIONAL BUENOS AIRES
SECRETARÍA DE EXTENSIÓN UNIVERSITARIA FRBA UTN

Centro de e-Learning

Centro de Formación, Investigación y Desarrollo de Soluciones de e-Learning.
UTN - FRBA. Secretaría de Cultura y Extensión Universitaria
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148 // e-learning@sceu.frba.utn.edu.ar

www.sceu.frba.utn.edu.ar/e-learning

Professional Testing Master

Presentación:

En esta primera Unidad del curso, explicaremos los conceptos y problemas fundamentales de software testing, así como también la importancia de cada una de las fases de la actividad usando un enfoque estratégico.

Unidad 1:

Fundamentos de testing

Objetivo:

Al terminar la Unidad los participantes:

- Se apropiarán de los principales conceptos relacionados con software testing y comprenderán los desafíos que impone la actividad.
- Habrán conocido y asimilado los principios y objetivos que guían la práctica de software testing.
- Conocerán las principales prácticas de testing llevadas a cabo en todo proyecto de software.

Contenido

1. Introducción a software testing
2. Importancia del testing
3. Aspectos psicológicos
4. Aspectos económicos
5. Objetivos y limitaciones del testing
6. Un enfoque estratégico

Introducción a software testing

El testing representa más del 50% del esfuerzo y el costo de los proyectos de desarrollo de software.

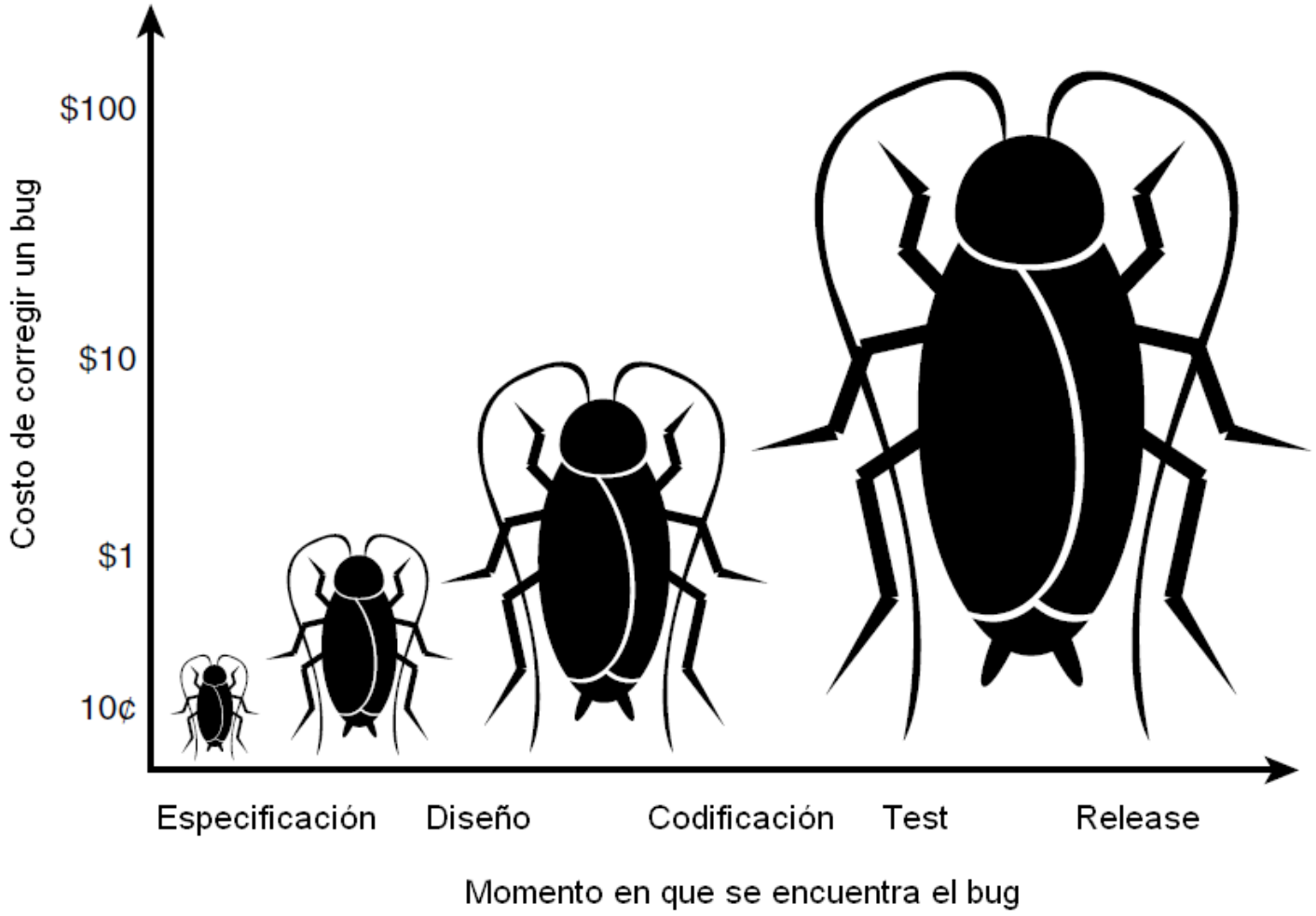
Sin embargo no se le presta la debida atención.

Esto causa pérdidas millonarias a las empresas.

Las Universidades no suelen ofrecer materias específicas de testing.

Es necesario adquirir los conceptos fundamentales y formarse en las mejores prácticas de la actividad.

Costo de los errores (bugs)



Costo de los errores (bugs)

Pérdida:
500 millones de dólares

Explosión del cohete Ariane 5 en 1996 por un bug en el software

Importancia del testing

La prueba de incluso un programa trivial no es una tarea fácil.

La situación empeora para sistemas más complejos y lenguajes orientados a objetos.

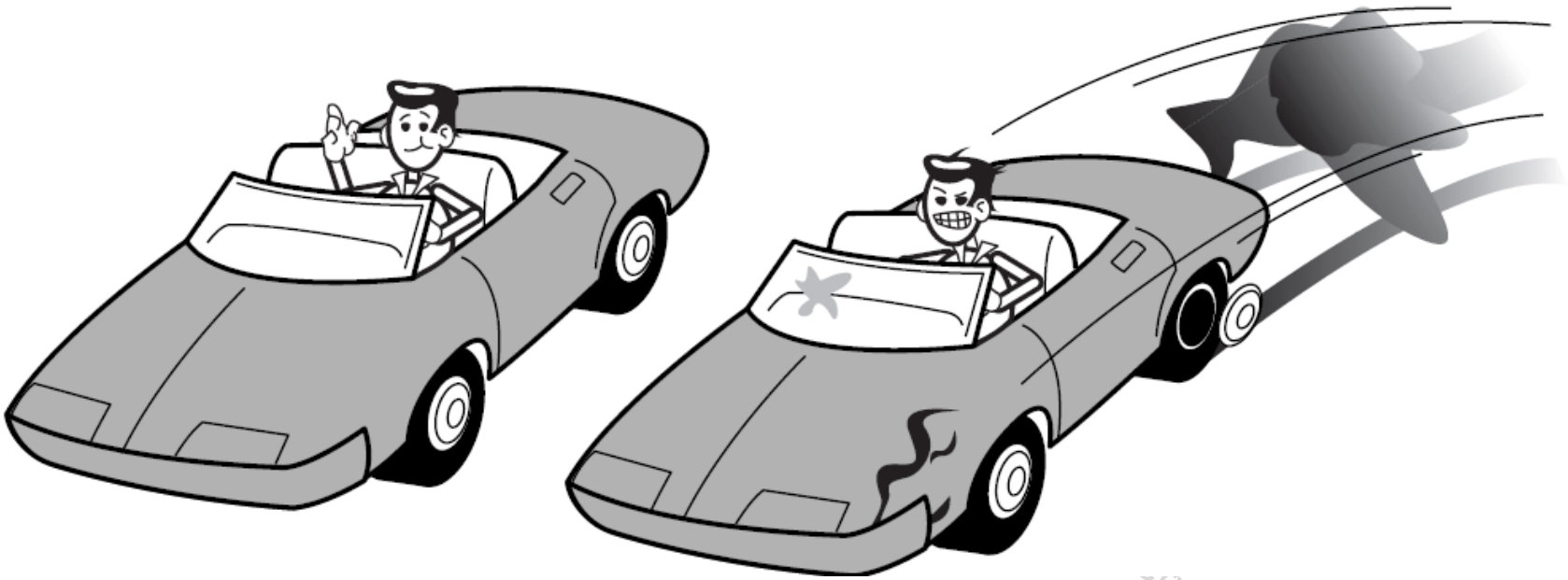
Aspectos psicológicos

Los aspectos psicológicos, tales como una definición incorrecta del testing son una de las causas del testing deficiente.

La prueba o testing es el proceso de ejecución de un programa con la intención de encontrar errores.

Un caso de prueba exitoso es aquel que promueve el progreso en encontrar errores, haciendo que el programa falle.

Aspectos psicológicos



Prueba para que no falle

Prueba para que falle

Aspectos económicos

No es posible probar exhaustivamente un programa para encontrar todos sus errores.

Por esto se debe recurrir a estrategias como por ejemplo las pruebas de caja negra y las pruebas de caja blanca.

Pruebas de caja negra

En las pruebas de caja negra los datos de prueba se derivan únicamente de las especificaciones, sin el conocimiento de la estructura interna del programa.

Debido a la imposibilidad de probar todas las entradas posibles (prueba exhaustiva) se deben hacer suposiciones razonables, pero no infalibles, sobre el programa (por ejemplo, si un programa clasifica 2,2,2 como un triángulo equilátero, hará lo mismo para 3,3,3)

Pruebas de caja blanca

Las pruebas de caja blanca, permiten examinar la estructura interna del programa. Esta estrategia deriva los datos de prueba de un análisis de la lógica del programa.

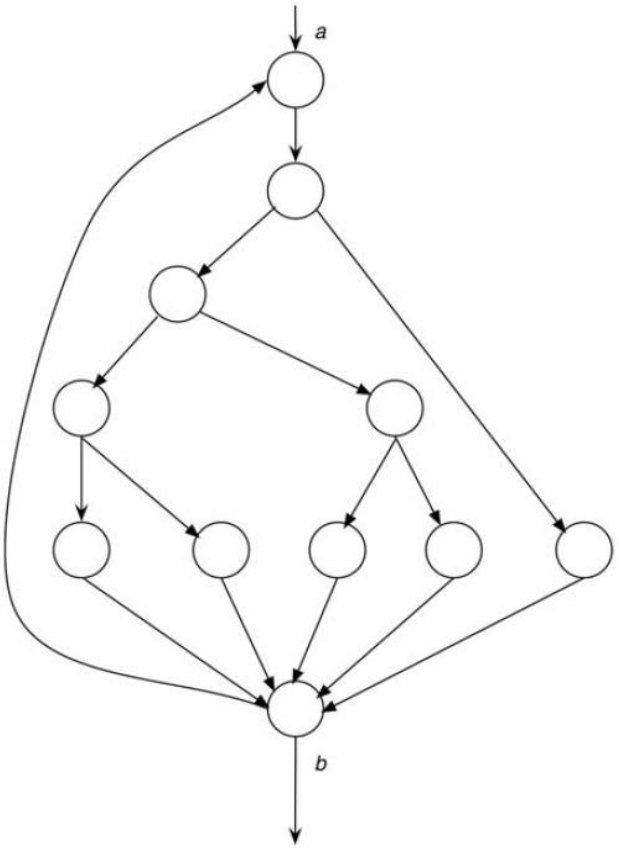
Si se ejecutan mediante de casos de prueba todos los posibles caminos (rutas) del flujo de control, entonces, posiblemente, el programa ha sido completamente probado.

Pruebas de caja blanca (cont.)

Sin embargo, la cantidad de posibles caminos (rutas) del flujo de control puede ser tan grande que sea inmanejable.

Por otro lado, aunque se pudieran probar todos los caminos, tampoco se detectarían todos los errores. Esto es debido a caminos faltantes en el programa, errores sensibles a los datos y errores en el cumplimiento de los requerimientos.

Pruebas de caja blanca (cont.)



Para este sencillo programa el número de rutas únicas es de 10^{14}

Objetivos y limitaciones del testing

El objetivo del testing es maximizar la cantidad de errores descubiertos dada una cierta cantidad de recursos invertidos en el test.

La primera prioridad del testing debe ser neutralizar los mayores riesgos de negocio (business killers).

La segunda prioridad son las actividades más frecuentes (regla del 80/20, el 20% que impulsa el 80% del negocio)

Un enfoque estratégico

El testing es un conjunto de actividades que se pueden planear anticipadamente y ejecutar como un proceso sistemático.

Una estrategia de testing debe contemplar tests de bajo nivel, necesarios para verificar que un pequeño fragmento de código fuente fue implementado correctamente, así como también tests de alto nivel para validar que la funcionalidad del sistema responde a los requerimientos del cliente.

Verificación y validación

Verificación: "¿Estamos construyendo el producto correctamente?“

Validación: "¿Estamos construyendo el producto correcto?"

Roles durante el testing

El desarrollador del software es el responsable de probar las unidades individuales (componentes) del programa.
El papel del grupo independiente de prueba (GIP) consiste en eliminar los problemas propios de dejar que el constructor (desarrollador) pruebe lo que él mismo ha construido.

Prueba unitaria

Tomando como guía la descripción del diseño a nivel de componentes, se prueban los caminos de control importantes.
La interfaz del módulo se prueba para asegurar que la información fluye apropiadamente hacia dentro y hacia afuera.
Se examinan las estructuras de datos locales para asegurar que los datos temporales mantienen la integridad durante todos los pasos de la ejecución.

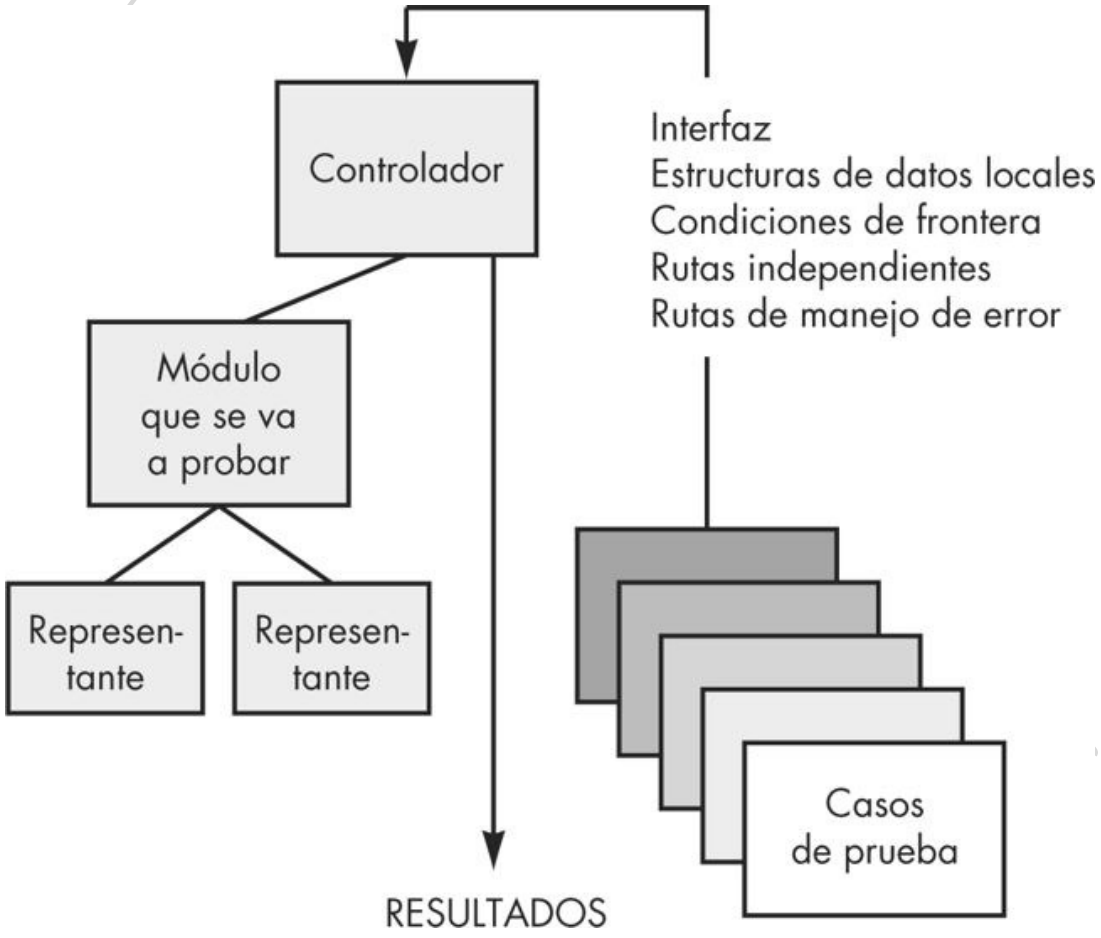
Prueba unitaria (cont.)

Se prueban las condiciones límite para asegurar que el módulo opera apropiadamente en los límites establecidos.
Se prueban todos los caminos de manejo de errores.

Procedimientos de prueba unitaria

Debido a que un componente no es un programa independiente, para cada prueba de unidad se debe desarrollar software controlador (driver) y/o de respaldo (stub) para reemplazar los componentes faltantes.

Procedimientos de prueba unitaria



Ambiente de prueba unitaria

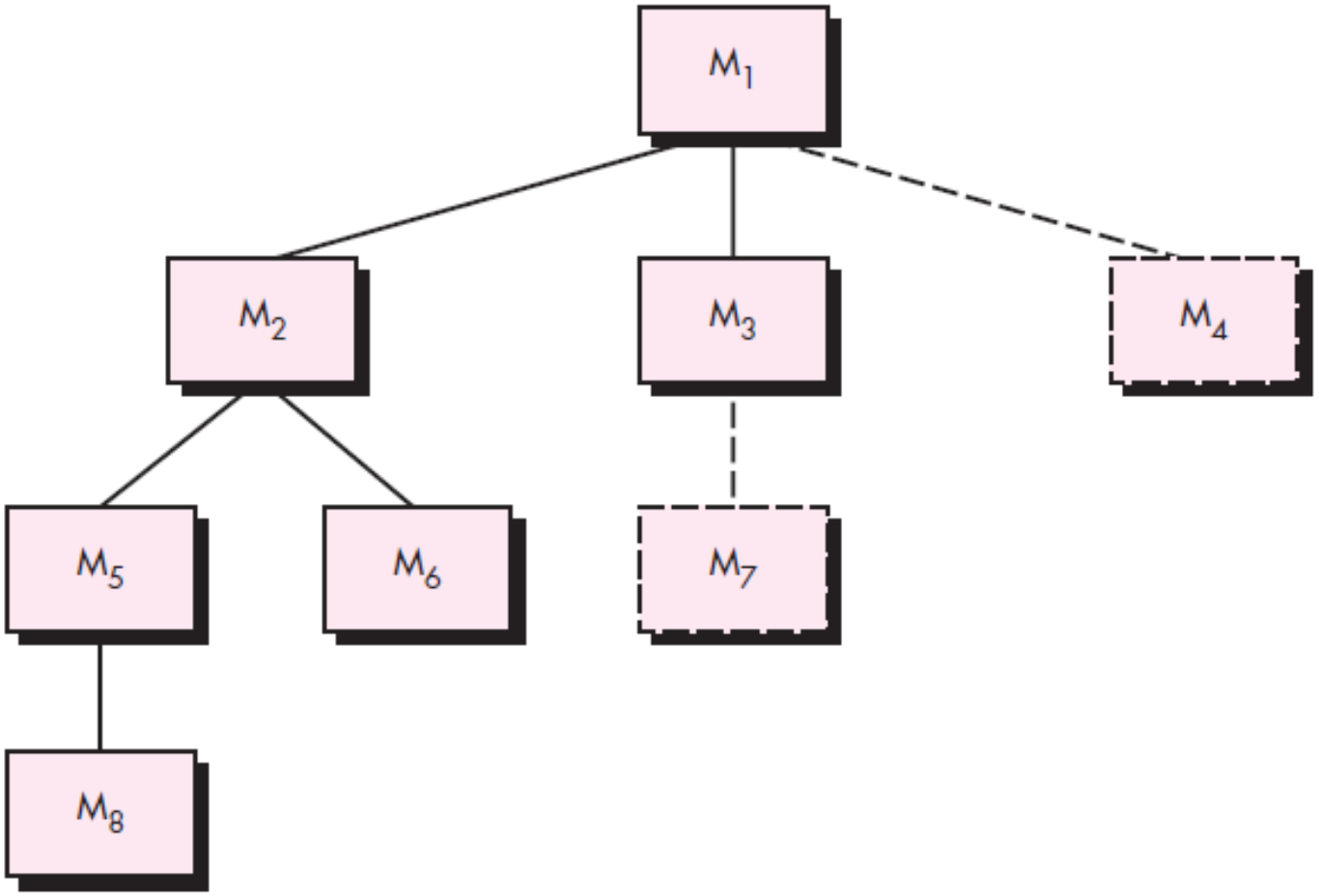
Prueba de integración

Aunque todos los componentes funcionen bien en forma individual, pueden surgir problemas al integrarlos. Esto se debe probar.
El enfoque “big bang” consiste en integrar todos los componentes al mismo tiempo y probarlos. Esto no es recomendable.
La alternativa es la integración incremental.

Integración descendente (dop-down)

Los módulos se integran al descender por la jerarquía de control, empezando con el módulo de control principal (programa principal).
Se utilizan stubs para reemplazar los módulos subordinados.
Se van reemplazando los stubs de a uno por los módulos subordinados de una de dos maneras: primero-en-profundidad o primero-en-anchura.
Cada vez que se reemplaza un stub se repiten las pruebas.

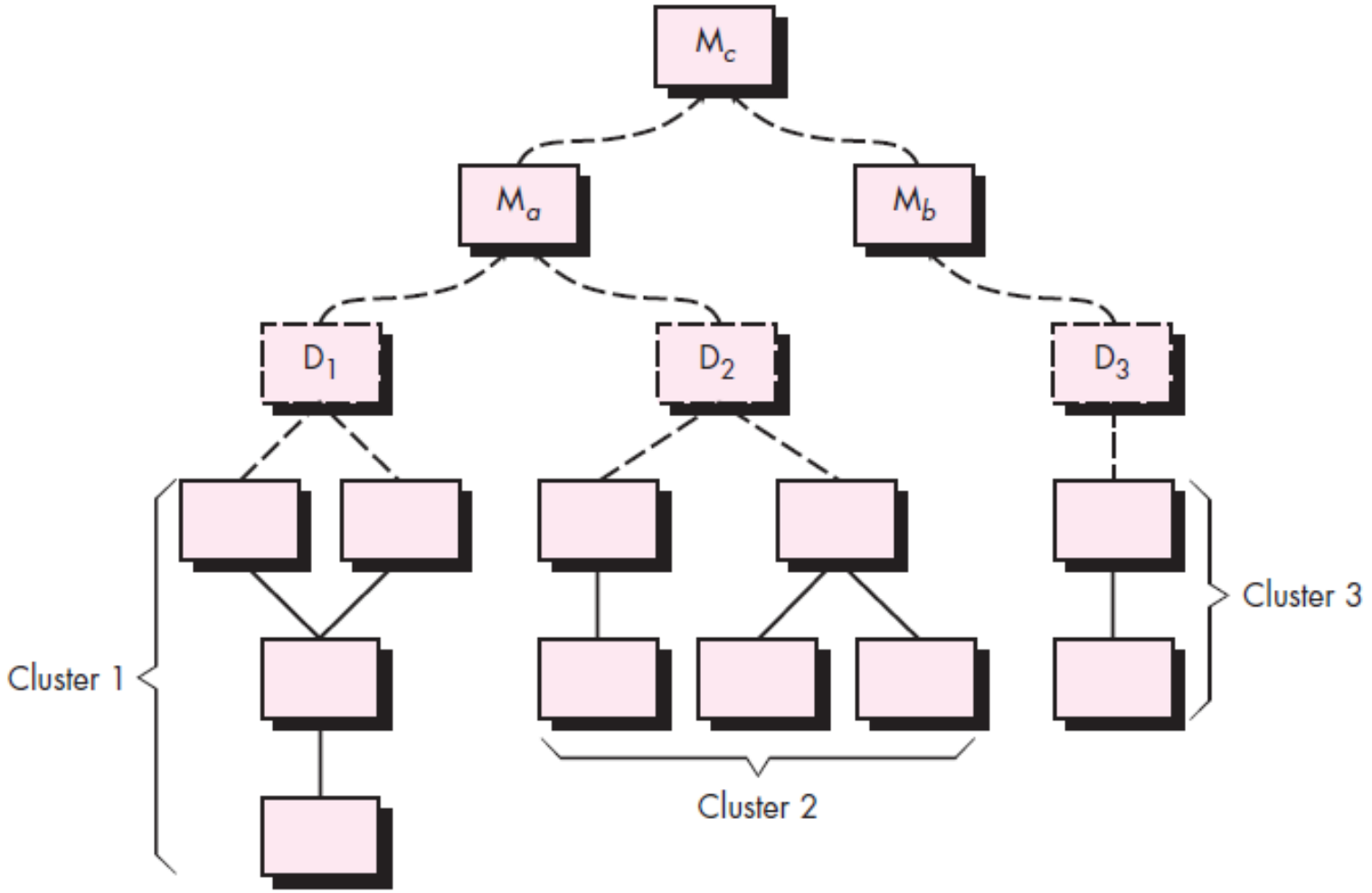
Integración descendente (dop-down)



Integración ascendente (bottom-up)

Se combinan los módulos de bajo nivel en grupos (clusters) que realicen una sub-función específica.
Se escribe un driver con el fin de coordinar la entrada y la salida de los casos de prueba.
Se prueba el grupo.
Se eliminan los drivers y se combinan los grupos ascendiendo por la estructura del programa.
Cada vez que se elimina un driver se repiten las pruebas.

Integración ascendente (bottom-up)



Prueba de validación

Consiste en comprobar que el software funciona de tal manera que satisface expectativas razonables del cliente.
Las expectativas razonables se definen en la especificación de requerimientos del software.

Prueba de sistema

El software sólo es un elemento de un sistema de cómputo más grande. Al final, el software se incorpora a elementos del sistema (como hardware, personas, información), y se realiza una serie de pruebas de integración del sistema y de validación. Este tipo de prueba incluye:

- Prueba de recuperación (recovery)
- Prueba de seguridad
- Prueba de resistencia (stress)
- Prueba de desempeño (performance)
- Prueba de despliegue (deployment)

Depuración (debugging)

Cuando un caso de prueba descubre un error, la depuración es la acción que lo elimina.
Las principales tácticas de depuración son:

- Depuración por fuerza bruta (el menos eficiente)
- Rastreo hacia atrás (back-tracking)
- Eliminación de causas

Estas tácticas se pueden complementar con herramientas automáticas y sobre todo con el aporte de un nuevo punto de vista de una tercera persona.

Consignas para discutir en el foro:

Reflexione a partir de las siguientes preguntas:

- ¿Por qué es importante el testing de un proyecto de software?
- ¿Cuáles son las dificultades que se enfrentan al encarar el testing?
- ¿Qué diferencia hay entre integración ascendente y descendente?
- Con sus palabras, describa la diferencia entre verificación y validación.
- ¿Quién debe realizar la prueba de validación: el desarrollador o el usuario del software? Justifique su respuesta.

Vierta sus respuestas en el foro y establezca un intercambio de opiniones con sus compañeros de curso.

Esperamos hayan disfrutado y aprovechado del estudio y las actividades propuestas en esta unidad!!!!!!!!!

UTN.BA
FACULTAD REGIONAL BUENOS AIRES
SECRETARÍA DE EXTENSIÓN UNIVERSITARIA FRBA UTN

Centro de e-Learning

Centro de Formación, Investigación y Desarrollo de Soluciones de e-Learning.
UTN - FRBA. Secretaría de Cultura y Extensión Universitaria
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148 // e-learning@sceu.frba.utn.edu.ar

www.sceu.frba.utn.edu.ar/e-learning