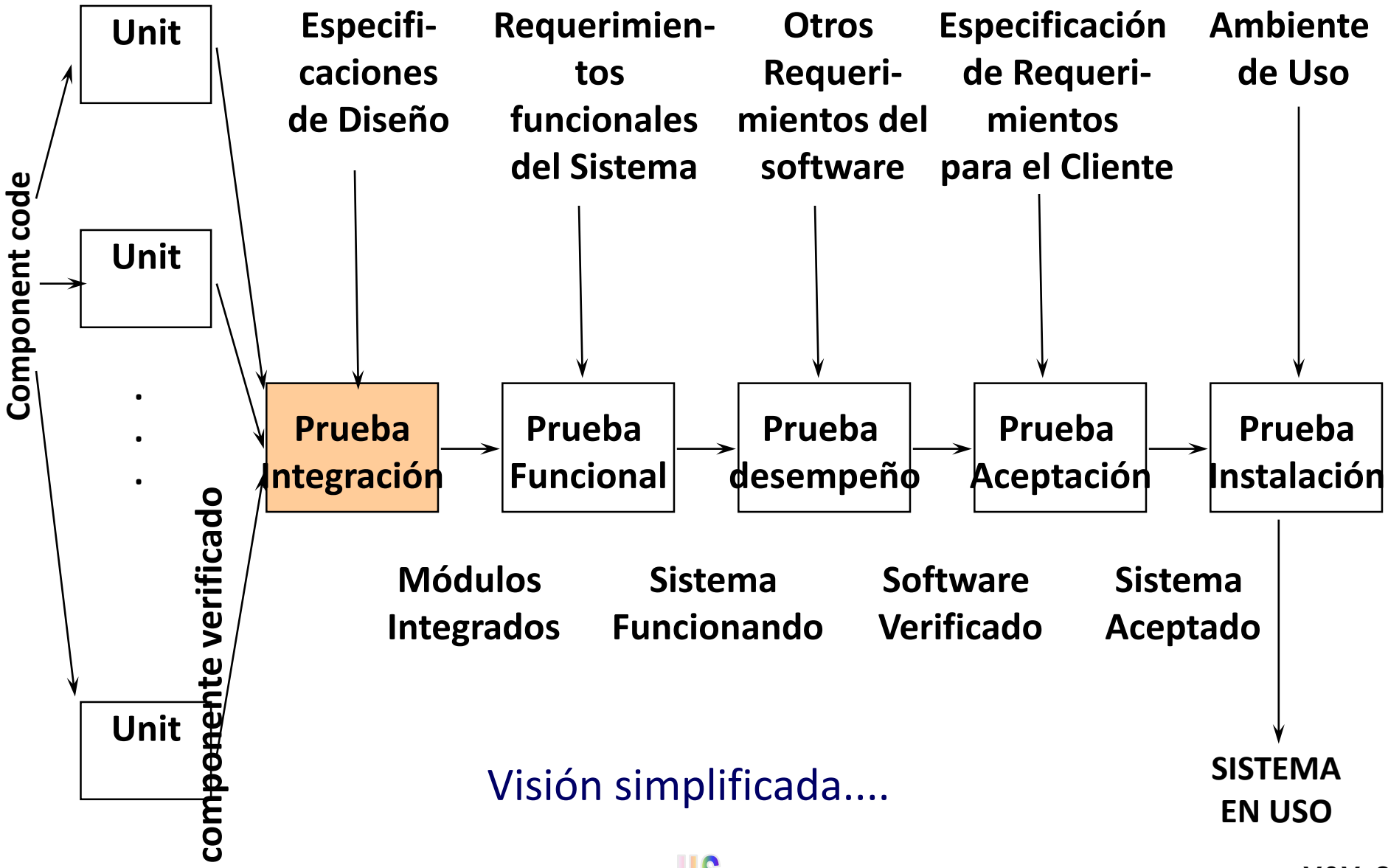
Verificación y Validación

Parte 2

Pruebas de Integración

- Temario
  - Pruebas de Módulos
  - Estrategias de Integración
    - Big-Bang
    - Bottom-Up
    - Top-Down
    - Sandwich
    - Por disponibilidad
  - Comparación de Estrategias
  - Builds en Microsoft

Proceso de V&V



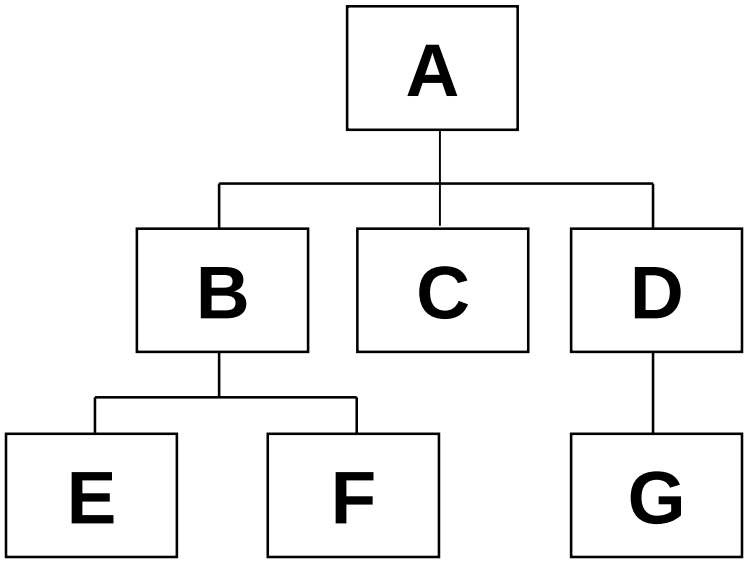
Visión simplificada....

Pruebas de Módulos

El módulo A “usa” a los módulos B, C, D.

El módulo B “usa” a los módulos E y F

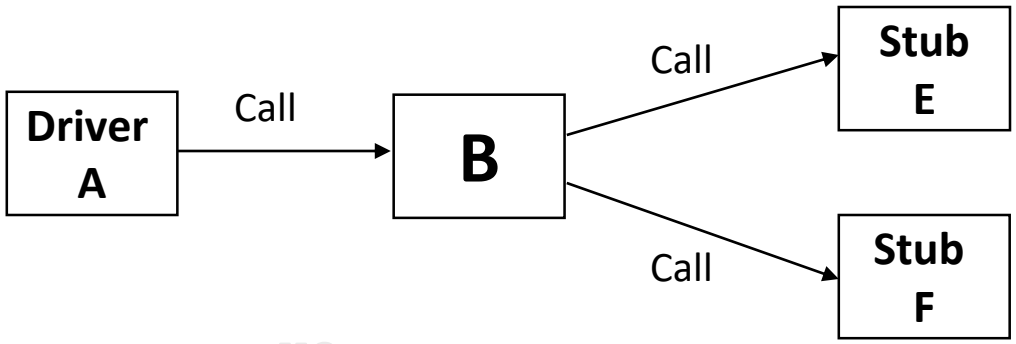
El módulo D “usa” al módulo G



Pruebas de Módulos

- Quiero probar al módulo B de forma aislada – No uso los módulos A, E y F
  - El módulo B es usado por el módulo A
    - Debo simular la llamada del modulo A al B – *Driver*
    - Normalmente el Driver es el que suministra los datos de los casos de prueba
  - El módulo B usa a los módulos E y F
    - Cuando llamo desde B a E o F debo simular la ejecución de estos módulos – *Stub*
  - Se prueba al módulo B con los métodos vistos en pruebas unitarias

Esto quiere decir definir criterios y generar test set que cubran esos criterios



Pruebas de Módulos

- Stub (simula la actividad del componente omitido)
  - Es una pieza de código con los mismos parámetros de entrada y salida que el módulo faltante pero con una alta simplificación del comportamiento. De todas maneras es costoso de realizar
  - Por ejemplo puede producir los resultados esperados leyendo de un archivo, o pidiéndole de forma interactiva a un testeador humano, o no hacer nada siempre y cuando esto sea aceptable para el módulo bajo test
  - Si el stub devuelve siempre los mismos valores al módulo que lo llama es probable que esté módulo no sea testeado adecuadamente. Ejemplo: el Stub E siempre devuelve el mismo valor cada vez que B lo llama → Es probable que B no sea testeado de forma adecuada

Pruebas de Módulos

- Driver
  - Pieza de código que simula el uso (por otro módulo) del módulo que está siendo testeado. Es menos costoso de realizar que un stub
  - Puede leer los datos necesarios para llamar al módulo bajo test desde un archivo, GUI, etc
  - Normalmente es el que suministra los casos de prueba al módulo que está siendo testeado

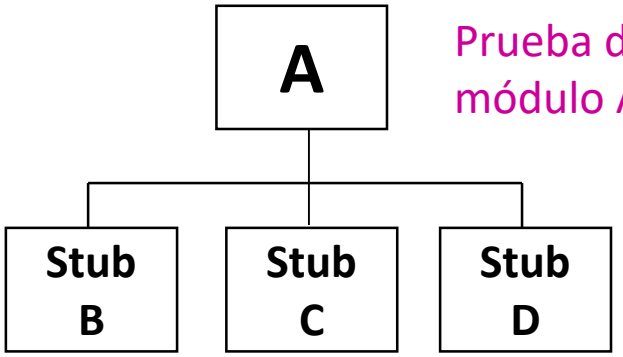
Estrategias de Integración

- No incremental
  - Big-Bang
- Incrementales
  - Bottom-Up (Ascendente)
  - Top-Down (Descendente)
  - Sandwich (Intercalada)
  - Por disponibilidad

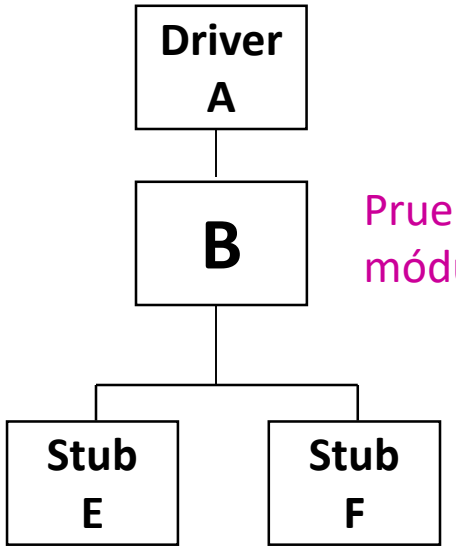
- El objetivo es lograr combinar módulos o componentes individuales para que trabajen correctamente de forma conjunta

Big-Bang

- Se prueba cada módulo de forma aislada y luego se prueba la combinación de todos los módulos a la vez



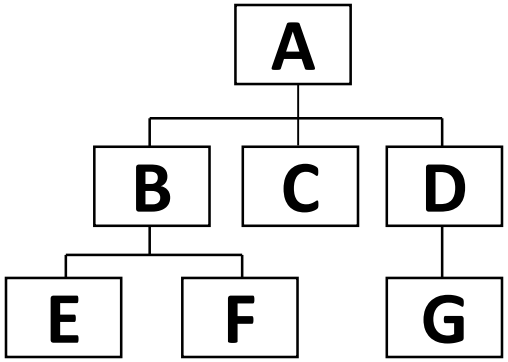
Prueba del módulo A



Prueba del módulo B

Se prueban los otros módulos de manera similar: C, D, E, F y G

No se hacen integraciones parciales



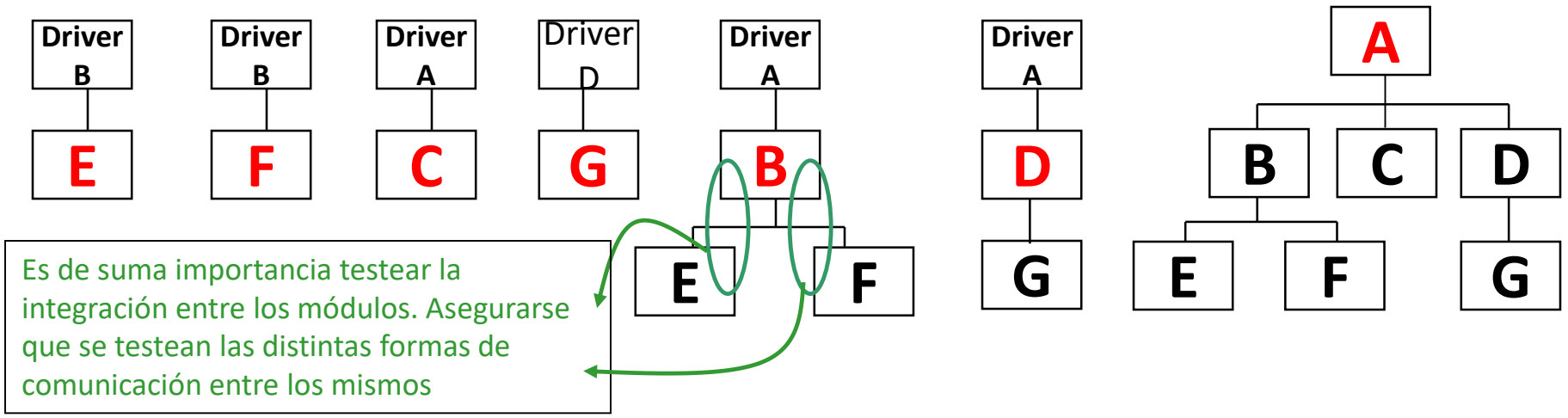
Después se integran todos los módulos a la vez y se prueba el funcionamiento en conjunto

Big-Bang

- Pros y contras
  - ✓ Existen buenas oportunidades para realizar actividades en paralelo (todos los módulos pueden ser probados a la vez)
  - ✗ Se necesitan stubs y drivers para cada uno de los módulos a ser testeados ya que cada uno se testea de forma individual
  - ✗ Es difícil ver cuál es el origen de la falla ya que integré todos los módulos a la vez
  - ✗ No se puede empezar la integración con pocos módulos
  - ✗ Los defectos entre las interfaces de los módulos no se pueden distinguir fácilmente de otros defectos
  - ✗ No es recomendable para sistemas grandes

Bottom-Up

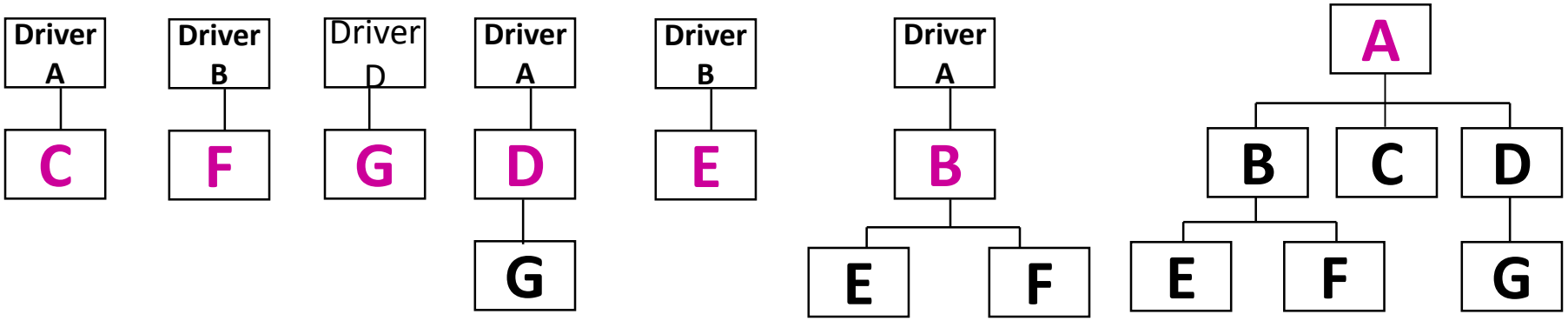
- Comienza por los módulos que no requieren de ningún otro para ejecutar
- Se sigue hacia arriba según la jerarquía “usa”
- Requiere de *Drivers* pero no de *Stubs*



- En los módulos marcados con color rojo se aplican métodos de prueba unitaria

Bottom-Up

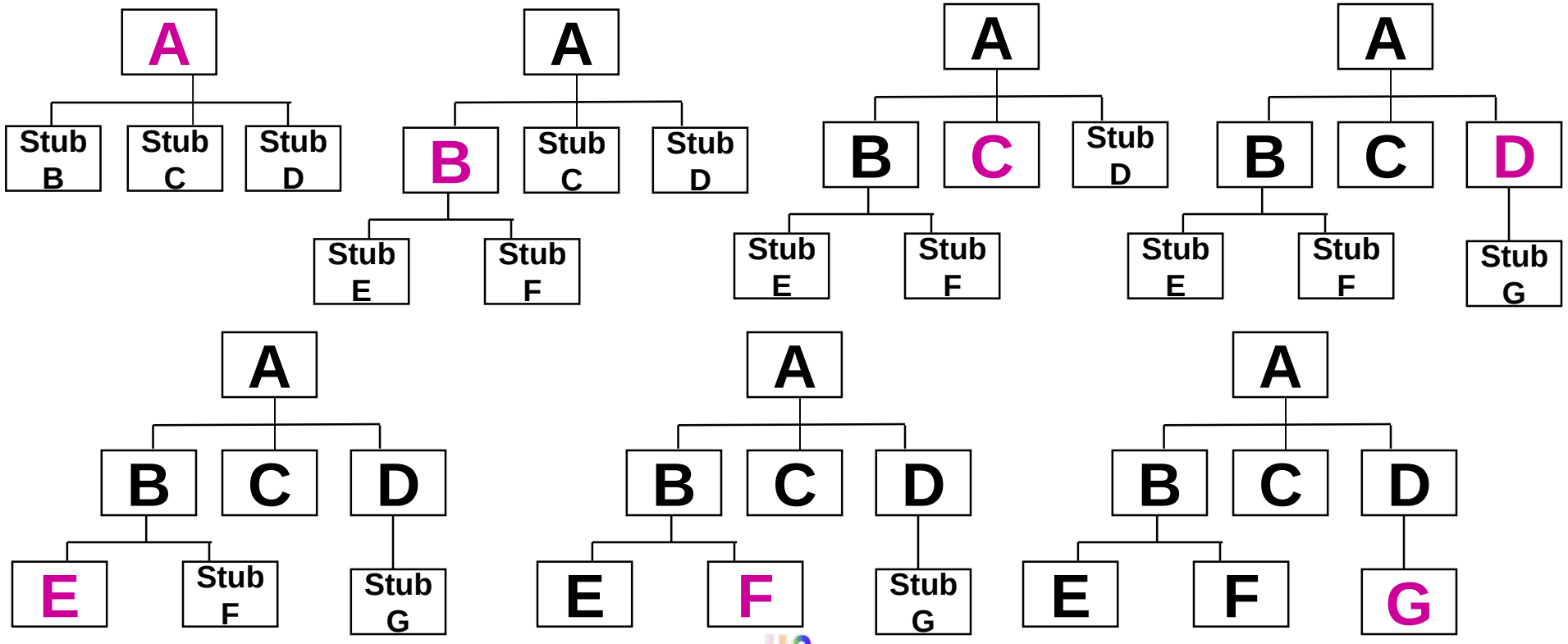
- La regla general indica que para probar un módulo todos los de “abajo” deben estar probados
- Supongamos que C, F y D son módulos críticos (módulo complicado, con un algoritmo nuevo o con posibilidad alta de contener errores, etc) entonces es bueno probarlos lo antes posible



- Hay que considerar que hay que diseñar e implementar de forma tal que los módulos críticos estén disponible lo antes posible. Es importante la planificación

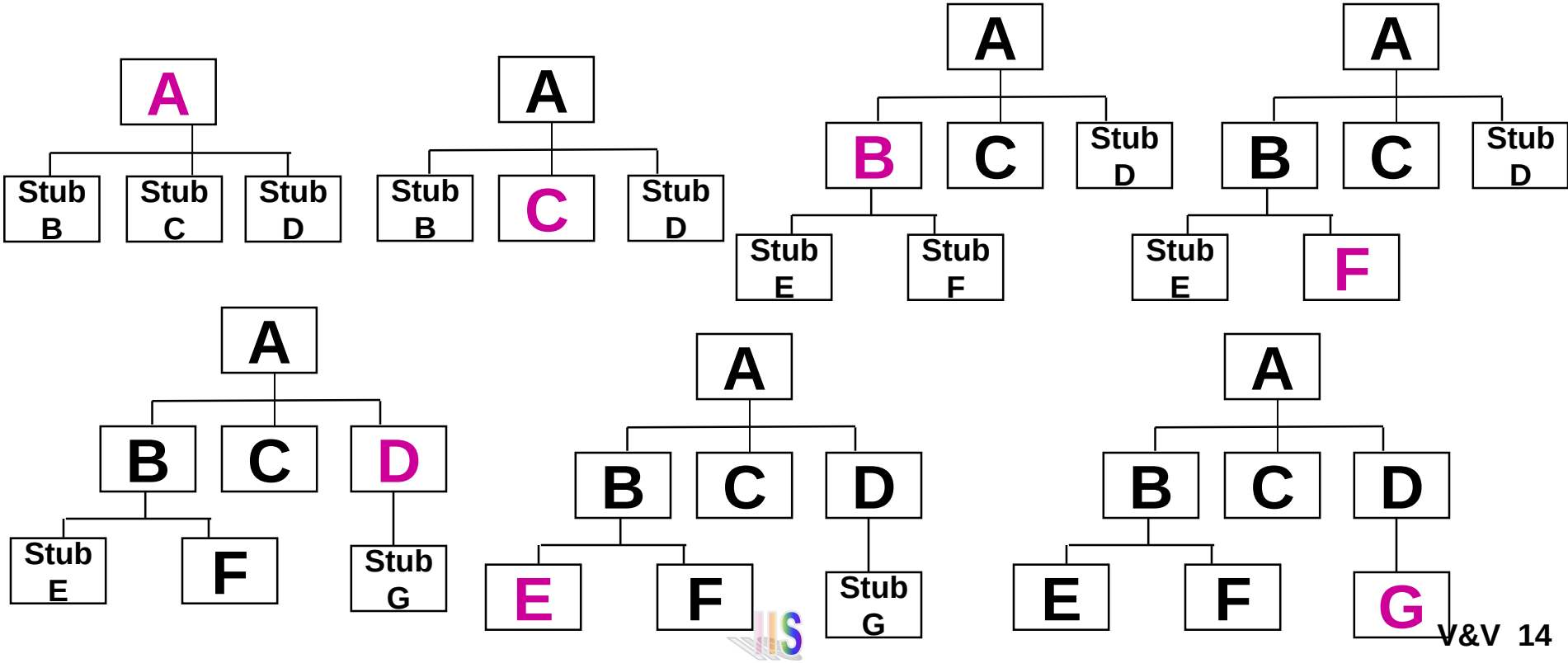
Top-Down

- Comienza por el módulo superior de la jerarquía usa
- Se sigue hacia abajo según la jerarquía “usa”
- Requiere de *Stubs* pero no de *Drivers*



Top-Down

- La regla general indica que para probar un módulo todos los de “arriba” (respecto al módulo a probar) deben estar probados
- Supongamos que C, F y D son módulos críticos entonces es bueno probarlos lo antes posible



Otras Técnicas

- Sandwich
  - Usa Top-Down y Bottom-Up. Busca probar los módulos más críticos primero
- Por Disponibilidad
  - Se integran los módulos a medida de que estos están disponibles

Comparación de las Estrategias

- No incremental respecto a incremental
  - Requiere mayor trabajo. Se deben codificar mas Stubs y Drivers que en las pruebas incrementales
  - Se detectan más tardíamente los errores de interfaces y/o suposiciones incorrectas entre los módulos ya que estos se integran al final
  - Es más difícil encontrar la falta que provocó la falla. En la prueba incremental es muy factible que el defecto esté asociado con el módulo recientemente integrado, o en interfaces con los otros módulos
  - Los módulos se prueban menos. En la incremental vuelvo a probar indirectamente a los módulos ya probados.

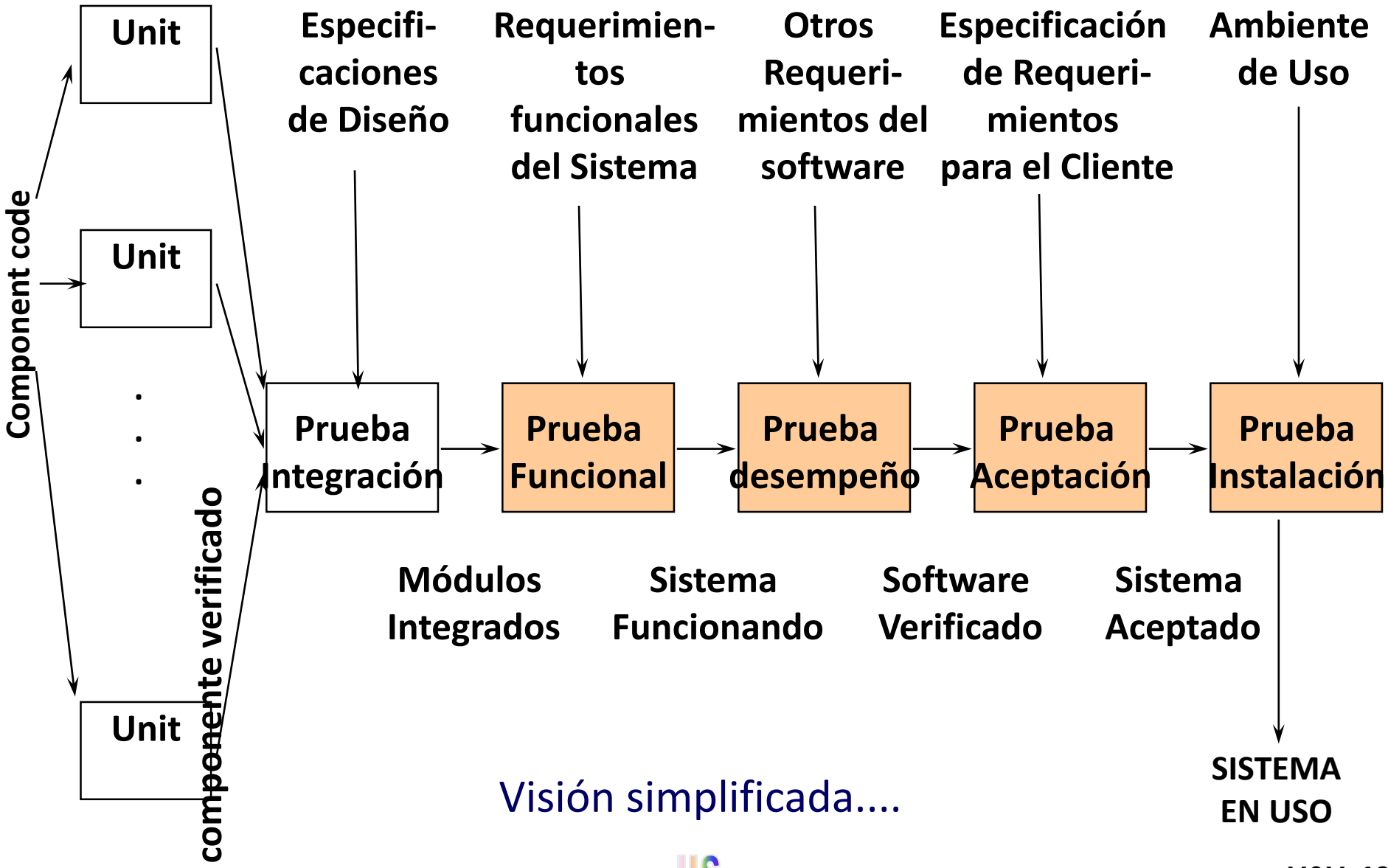
Comparación de las Estrategias

- Top-down (focalizando en OO)
  - ✓ Resulta fácil la representación de casos de prueba reales ya que los módulos superiores tienden a ser GUI
  - ✓ Permite hacer demostraciones tempranas del producto
  - ✗ Los stubs resultan muy complejos
  - ✗ Las condiciones de prueba pueden ser muy difíciles de crear
- Bottom-Up (focalizando en OO)
  - ✓ Las condiciones de las pruebas son más fáciles de crear
  - ✓ Al no usar stubs se simplifican las pruebas
  - ✗ El programa como entidad no existe hasta no ser agregado el último módulo

Pruebas del Sistema

- Temario
  - Prueba Funcional
    - A partir de casos de uso
  - Prueba de Desempeño
  - Prueba de Aceptación
  - Prueba de Instalación

Proceso de V&V



Visión simplificada....

Las Distintas Pruebas del Sist.

- Prueba de función
  - Verifica que el sistema integrado realiza sus funciones especificadas en los requerimientos
- Prueba de desempeño
  - Verifica los requerimientos no funcionales del sistema
- Prueba de aceptación
  - Se realiza junto con el cliente. Busca asegurar que el sistema es el que el cliente quiere
- Prueba de instalación
  - Se realizan las pruebas en el ambiente objetivo

Prueba de Función

- La idea es probar las distintas funcionalidades del sistema
  - Se prueba cada funcionalidad de forma individual
    - Esto puede ser testeo de casos de uso
  - Se prueban distintas combinaciones de funcionalidades
    - Ciclos de vida de entidades (alta cliente, modificación del cliente, baja del cliente)
    - Procesos de la organización (pedido de compra, gestión de la compra, envió y actualización de stock)
    - Esto se puede ver como testeo de ciclos de casos de usos
- El enfoque es más bien de caja negra
- Está prueba está basada en los requerimientos funcionales del sistema

Prueba de Función – Use Case

- Se hace la prueba funcional a partir de los casos de uso
- Se identifican los distintos escenarios posibles y se usan como base para las condiciones de prueba
- Se completan las condiciones en función de los tipos de datos de entrada y de salida
- A partir de las condiciones se crean los Casos de Prueba

Caso de Uso: Retiro

Precondición: el cliente ingresó una tarjeta válida e ingresó nro. de PIN

Flujo Principal:

1. El CA despliega las distintas alternativas disponibles,el Cliente elige Retiro
2. El CA pide cuenta y monto y el Cliente los elige (o ingresa)
3. CA envía Id. Tarjeta, PIN, cuenta y monto
4. SC (Servicio de Cajeros) contesta: Continuar (OK) o No Continuar
5. CA dispensa el dinero
6. CA devuelve la tarjeta
7. CA imprime el recibo

4A. Importe inválido

Si el monto indicado por el cliente no puede obtenerse a partir de los billetes de que dispone el CA, este despliega un mensaje de advertencia y le solicita que ingrese nuevamente el importe. No hay límite de repeticiones.

4B. No hay suficiente saldo en la cuenta.

4B1. CA despliega mensaje al Cliente y devuelve la tarjeta (no se imprime recibo)

Caso de Uso: Retiro

Instancias de un Casos de Uso

1. Flujo Principal
2. Flujo Principal- Importe inválido (n)
3. Flujo Principal- No hay suficiente saldo en cuenta

Condiciones de Prueba

Condiciones	Tipo de Resultado	Caso
1. Normal	Retiro exitoso	
1.1 Normal Caja de Ahorros		
1.2 Normal Cuenta Corriente		
2. Normal-Importe inválido (n)	Retiro exitoso	
3. Normal- sin saldo	No exitoso	

Casos de Prueba

\Caso	A1	A2	A3		A4
Entradas					
Tarjeta	13131	13131	13131		23232
PIN	9001	9001	9001		9002
Cuenta	CA128	CC321	CC321		CA228
Monto	100	500	12	1200	100
Salidas			Importe inválido		Sin saldo
Dispensa	$100	$500		$1200	
Dev. tarjeta	Sí	Sí		Sí	Sí
Recibo	13131...	13131...		13131...	
Condics.	1.1	1.2		2	3

Prueba de Desempeño

- Lo esencial es definir
  - Procedimientos de prueba
    - Ejemplo: Tiempo de respuesta
      - Simular la carga, tomar los tiempos de tal manera, número de casos a probar, etc
  - Criterios de aceptación
    - Ejemplo
      - El cliente lo valida si en el 90% de los casos de prueba en el ambiente de producción se cumple con los requerimientos de tiempo de respuesta
  - Características del ambiente
    - Definir las características del ambiente de producción ya que estas hacen grandes diferencias en los requerimientos no funcionales

Prueba de Desempeño

- Prueba de estrés (esfuerzo)
- Prueba de volumen
- Prueba de facilidad de uso
- Prueba de seguridad
- Prueba de rendimiento
- Prueba de configuración
- Prueba de compatibilidad
- Prueba de facilidad de instalación
- Prueba de recuperación
- Prueba de confiabilidad
- Prueba de documentación

Prueba de Desempeño

- Prueba de estrés (esfuerzo)
  - Esta prueba implica someter al sistema a grandes cargas o esfuerzos considerables.
  - No confundir con las pruebas de volumen. Un esfuerzo grande es un pico de volúmenes de datos (normalmente por encima de sus límites) en un *corto período de tiempo*
  - No solo volúmenes de datos sino también de usuarios, dispositivos, etc
  - Analogía con dactilógrafo
    - Volumen: Ver si puede escribir un documento enorme
    - Esfuerzo: Ver si puede escribir a razón de 50 palabras x minuto

Prueba de Desempeño

- Prueba de volumen
  - Esta prueba implica someter al sistema a grandes volúmenes de datos
  - El propósito es mostrar que el sistema no puede manejar el volumen de datos especificado

Prueba de Desempeño

- Prueba de facilidad de uso
  - Trata de encontrar problemas en la facilidad de uso
  - Algunas consideraciones a tener en cuenta
    - ¿Ha sido ajustada cada interfaz de usuario al nivel educacional del usuario final y a las presiones del ambiente?
    - ¿Son las salidas del programa significativas y desprovistas de la “jerga de las computadoras”?
    - ¿Son directos los diagnósticos de error (mensajes de error), o requieren de un doctor en ciencias de la computación?
  - Lo que importa es darse cuenta lo necesario de este tipo de pruebas. Si no se realizan el sistema puede ser inutilizable.
  - Algunas se pueden realizar durante el prototipado

Prueba de Desempeño

- Prueba seguridad
  - La idea es tratar de generar casos de prueba que burlen los controles de seguridad del sistema
  - Ejemplo
    - Diseñar casos de prueba para vencer el mecanismo de protección de la memoria de un sistema operativo
  - Una forma de encontrar casos de prueba es estudiar problemas conocidos de seguridad en sistemas similares. Luego mostrar la existencia de problemas parecidos en el sistema bajo test.
  - Existen listas de descripciones de fallas de seguridad en diversos tipos sistemas

Prueba de Desempeño

- Prueba de rendimiento
  - Generar casos de prueba para mostrar que el sistema no cumple con sus especificaciones de rendimiento
  - Casos
    - Tiempos de respuesta en sistemas interactivos
    - Tiempo máximo en ejecución de alguna tarea
  - Normalmente esto está especificado bajo ciertas condiciones de carga y de configuración del sistema
    - Ejemplo
      - El proceso de facturación no debe durar más de una hora en las siguientes condiciones:
        - » Se ejecuta en el ambiente descrito en el anexo 1
        - » Se ejecuta para 10.000 empleados

Prueba de Desempeño

- Prueba de configuración
  - Se prueba el sistema en distintas configuraciones de hardware y software
  - Como mínimo las pruebas deben ser realizadas con las configuraciones mínima y máxima posibles

- Prueba de compatibilidad
  - Son necesarias cuando el sistema bajo test tiene interfaces con otros sistemas
  - Se trata de determinar que las funciones con la interfaz no se realizan según especifican los requerimientos

Prueba de Desempeño

- Prueba de facilidad de instalación
  - Se prueban las distintas formas de instalar el sistema
- Prueba de recuperación
  - Se intenta probar que no se cumplen los requerimientos de recuperación
  - Para esto se simulan o crean fallas y luego se testea la recuperación del sistema

Prueba de Desempeño

- Prueba de documentación
  - La documentación del usuario debe ser objeto de inspección controlando su exactitud y claridad (entre otros)
  - Además todos los ejemplos que aparezcan en la misma deben ser codificados como casos de prueba. Es importantísimo que al ejecutar estos casos de prueba los resultados sean los esperados
    - Un sistema uruguayo tiene en su manual de usuario un ejemplo que no funciona en el sistema

Prueba de Aceptación e Instalación

- Prueba de aceptación
  - El cliente realiza estas pruebas para ver si el sistema funciona de acuerdo a sus requerimientos y necesidades

- Prueba de instalación
  - Su mayor propósito es encontrar errores de instalación
  - Es de mayor importancia si la prueba de aceptación no se realizó en el ambiente de instalación

Otras Pruebas

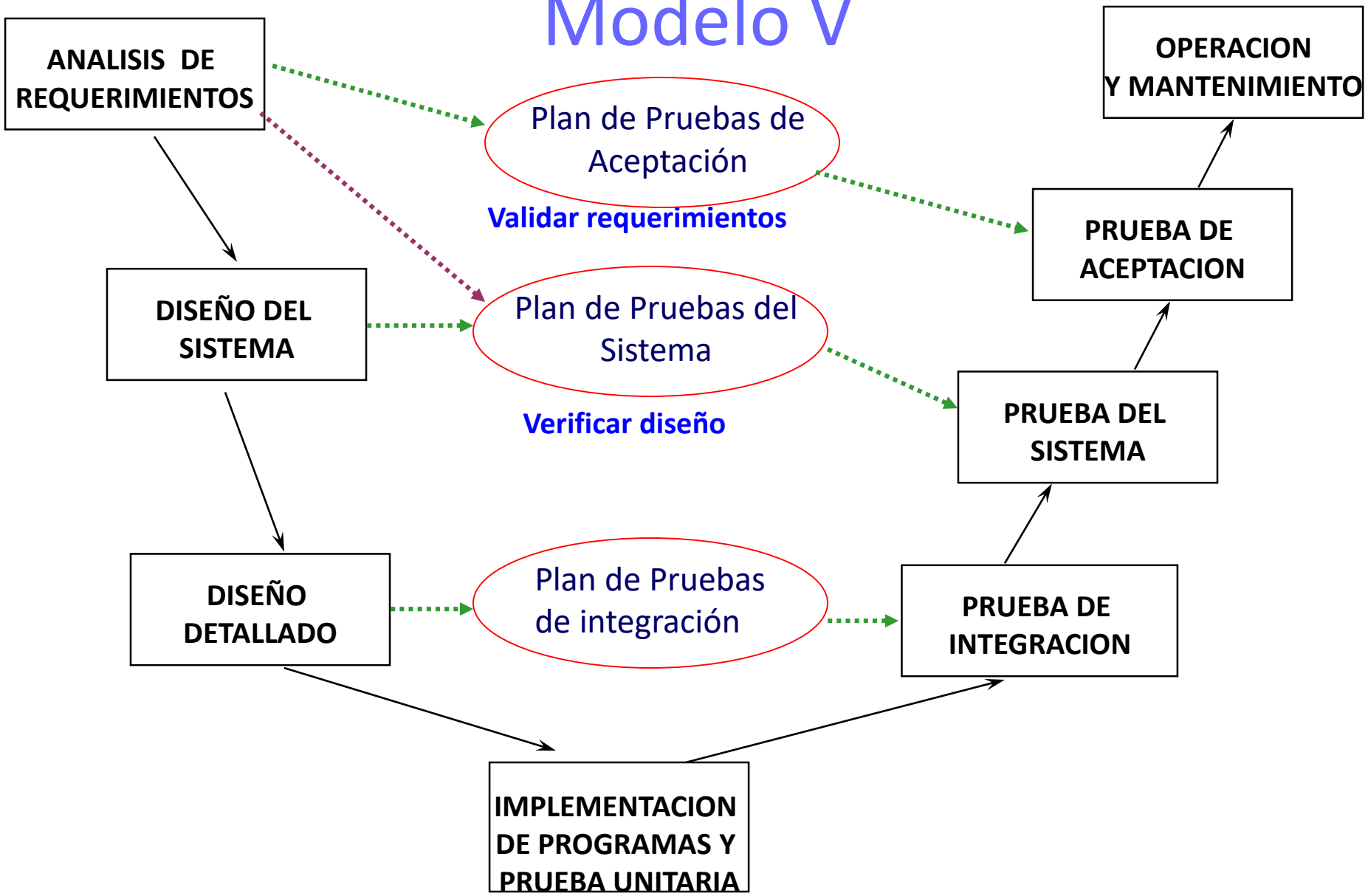
- Desarrollo para múltiples clientes
  - Prueba Alfa
    - Realizada por el equipo de desarrollo sobre el producto final
  - Prueba Beta
    - Realizada por clientes elegidos sobre el producto final
    - Se podría decir que Windows está siempre en Beta testing
- Desarrollo de un sistema que sustituye a otro
  - Pruebas en paralelo
    - Se deja funcionando el sistema “viejo” mientras se empieza a usar al nuevo
    - Se hacen comparaciones de resultados y se intenta ver que el sistema nuevo funciona adecuadamente

Otras Pruebas

- Pruebas de regresión
  - Después de que se realizan modificaciones se verifica que lo que ya estaba probado sigue funcionando
  - Conviene tener automatizadas las pruebas así las pruebas de regresión se pueden hacer con mucha mayor rapidez
- Pruebas piloto
  - Se pone a funcionar el sistema en producción de forma localizada
  - Menor impacto de las fallas

Planificación de V&V

Modelo V


ANALISIS DE REQUERIMIENTOS
Plan de Pruebas de Aceptación
Validar requerimientos
OPERACION Y MANTENIMIENTO
PRUEBA DE ACEPTACION
DISEÑO DEL SISTEMA
Plan de Pruebas del Sistema
Verificar diseño
PRUEBA DEL SISTEMA
DISEÑO DETALLADO
Plan de Pruebas de integración
PRUEBA DE INTEGRACION
IMPLEMENTACION DE PROGRAMAS Y PRUEBA UNITARIA

Generalidades

El mayor error cometido en la planificación de un proceso de prueba:

Suponer al momento de realizar el cronograma, que no se encontrarán fallas

Los resultados de esta equivocación son obvios

- Se subestima la cantidad de personal a asignar
- Se subestima el tiempo de calendario a asignar
- Entregamos el sistema fuera de fecha o ni siquiera entregamos

Generalidades

- La V&V es un proceso caro → se requiere llevar una planificación cuidadosa para obtener el máximo provecho de las revisiones y las pruebas para controlar los costos del proceso de V&V
- El proceso de planificación de V&V
  - Pone en la balanza los enfoques estático y dinámico para la verificación y la validación
  - Utiliza estándares y procedimientos para las revisiones y las pruebas de software
  - Establece listas de verificación para las inspecciones
  - Define el plan de pruebas de software
- A sistemas más críticos mayor verificación estática

Planes de Prueba (Sommerville)

- Comprende aplicar los estándares para el proceso de prueba más que a describir las pruebas del producto
- Permite que el personal técnico obtenga una visión global de las pruebas del sistema y ubique su propio trabajo en ese contexto
- También proveen información al personal responsable de asegurar que los recursos de hardware y software apropiados estén disponibles para el equipo de pruebas

Planes de Prueba (Sommerville)

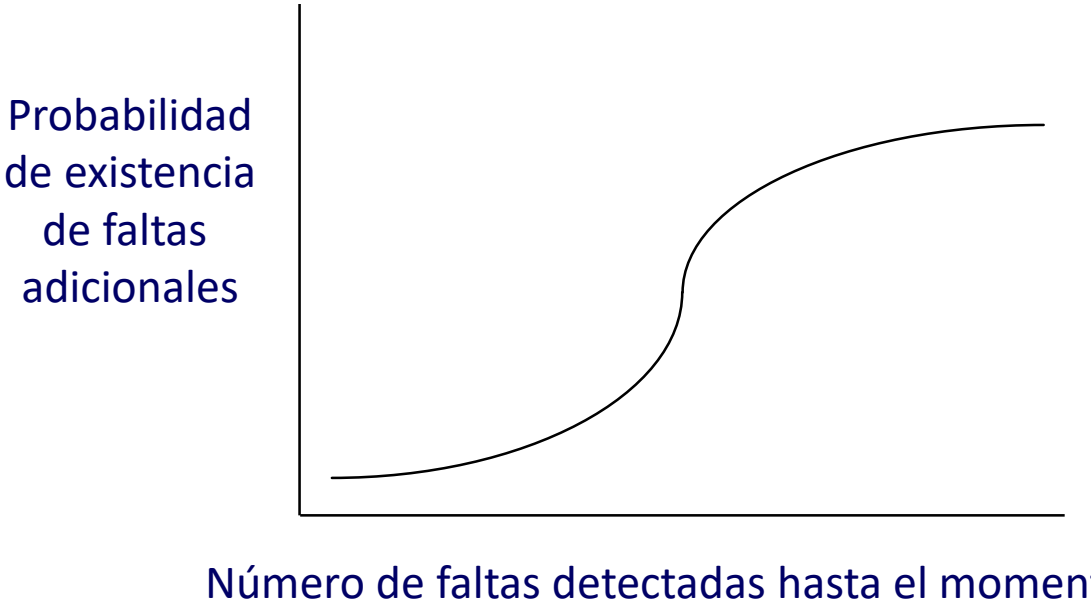
- Componentes principales del plan
  - Proceso de prueba
    - Descripción principal de las fases de prueba
  - Requerimientos de rastreo o seguimiento
    - Se planifican pruebas de tal forma que se puedan testear individualmente todos los requerimientos
  - Elementos probados
    - Se especifican los productos del proceso de software a probar
  - Calendarización del las pruebas
    - Calendarización de todas las pruebas y asignación de recursos. Esto se vincula con el calendario general del proyecto

Planes de Prueba (Sommerville)

- Componentes principales del plan (2)
  - Procedimiento de registro de las pruebas
    - Los resultados de la ejecución de las pruebas se deben grabar sistemáticamente. Debe ser posible auditar los procesos de prueba para verificar que se han llevado a cabo correctamente
  - Requerimientos de hardware y software
    - Esta sección señala el hardware y software requerido y la utilización estimada del hardware
  - Restricciones
    - Se anticipan las restricciones que afectan al proceso de prueba, como por ejemplo la escasez de personal
- Como otros planes, el plan de pruebas no es estático y debe revisarse regularmente

Terminación de la Prueba

Más Faltas… más por Encontrar



Estudio de Myers en 1979

- Va en contra de nuestra intuición
- Hace difícil saber cuando detener la prueba
- Sería bueno estimar el número de defectos remanente
  - Ayuda a saber cuando detener las pruebas
  - Ayuda a determinar el grado de confianza en el producto

Criterios de Terminación

- Criterios usados pero contraproducentes
  - Terminar la prueba cuando
    1. **El tiempo establecido para la misma ha expirado**
       - Es inútil puesto que puede ser satisfecho sin haber hecho absolutamente nada. Este criterio no mide la calidad de las pruebas realizadas
    2. **Todos los casos de prueba se han ejecutado sin detectar fallas**
       - Es inútil porque es independiente de la calidad de los casos de prueba. Es también contraproducente porque subconscientemente se tienden a crear casos de prueba con baja probabilidad de descubrir errores

Categorías de Criterios (Myers)

1. Por criterios del diseño de casos de prueba
   - Basa la terminación de las pruebas en el uso de métodos específicos de casos de prueba (ejemplo: cubrimiento de arcos)
2. Detención basada en la cantidad de defectos
   - Ejemplos
     - Detectar 3 defectos por módulo y 70 en la prueba funcional
     - Tener detectado el 90% de los defectos del sistema
3. Basada en la cantidad de fallas detectadas por unidad de tiempo, durante las pruebas

Por Criterios del Diseño de Casos de Prueba

- Se definen criterios de terminación según el cumplimiento de un criterio de cubrimiento y que todos esos casos de prueba se ejecuten sin provocar fallas
- Ejemplo:
  - Termino las pruebas cuando ejecute todas las trayectorias independientes del programa y todos los test den igual al resultado esperado
  - Esto se puede hacer con ayuda de herramientas que indican qué cobertura tengo del código con los casos de prueba ejecutados

Detención Basada en la Cantidad de Defectos

- Termino la prueba basado en la cantidad de defectos → ¿cómo determino el número de defectos a ser detectado para dar por terminada la prueba?
  - Necesito estimar el número total de defectos. De esta manera puedo saber cuando parar la prueba
  - Ejemplo de criterio
    - Detener la prueba cuando haya detectado el 90% de los defectos
    - Para saber cuando detecte el 90% de los defectos uso la estimación del total de defectos
- A continuación se presentan formas de estimar la cantidad total de defectos

Estimar la cantidad de defecto
Siembra de Defectos

- Sirve para estimar el número de defectos
  - Se agregan defectos intencionalmente en un módulo
  - Se hacen pruebas para detectar defectos en ese módulo
  - Se usa la siguiente relación

$$\frac{\text{Defectos sembrados detectados}}{\text{Total de defectos sembrados}} = \frac{\text{Defectos no sembrados detectados}}{\text{Total de defectos no sembrados}}$$

  - Despejando se obtiene una estimación del total de defectos no sembrados
  - ✓ Es un enfoque simple y útil
  - ✗ Asume que los defectos sembrados son representativos de los defectos reales (tipo y complejidad)

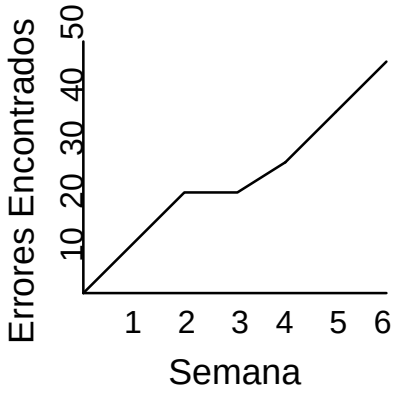
Estimar la cantidad de defecto
Pruebas Independientes

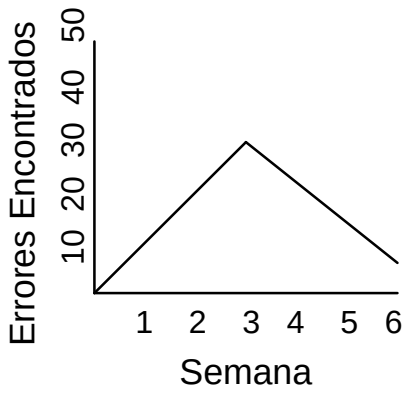
- Pongo dos grupos de prueba a probar el mismo programa
- Defectos encontrados por Grupo 1 = **x**
- Defectos encontrados por Grupo 2 = **y**
- Defectos en común = **q** entonces **q<=x & q<=y**
- Defectos totales = **n** (número a estimar)
- Efectividad = **cantDefectosEncontrados / n**
- Efectividad Grupo1 = **E1 = x/n** **E2 = y/n**
- Consideremos el Grupo 1 y tomemos que la tasa de efectividad es igual para cualquier parte del programa
  - El Grupo 1 encontró **q** de los **y** defectos detectados por el grupo 2

  → **E1 = q / y**
- Entonces se deriva la siguiente formula **n = (x*y)/q**

Detención Basada en Fallas por Unidad de Tiempo

- Se registra el número de fallas que se detectan durante la prueba por unidad de tiempo. Luego se grafican estos valores
- Examinando la forma de la curva se puede determinar cuando detener las pruebas





- Si tengo la gráfica de la izquierda y estoy usando este método de detención debería continuar las pruebas
- La gráfica de la derecha podría estar indicando que es un buen momento para parar las pruebas. De todos modos hay que garantizar que la bajada en la detección de fallas no se deba a factores como:
  - Menos cantidad de pruebas ejecutadas
  - Agotamiento de los casos de prueba preparados

Una Combinación

- Para la fase de prueba de sistema la mezcla de las dos últimas categorías puede resultar positiva

Se pararán las pruebas cuando se detecte un número predeterminado de defectos o cuando haya transcurrido el tiempo fijado para ello, eligiendo la que ocurra más tarde, siempre que un análisis del gráfico del número de defectos detectados por unidad de tiempo en función del tiempo indique que la prueba se ha tornado improductiva