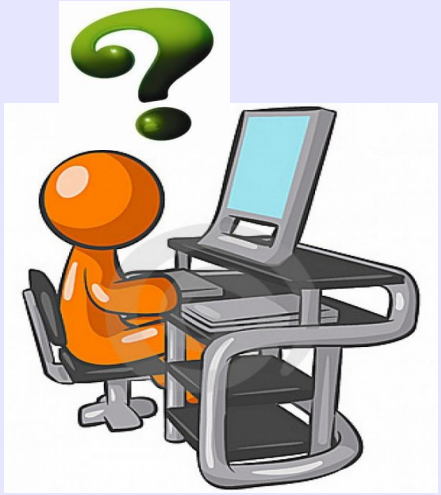
Tema 1

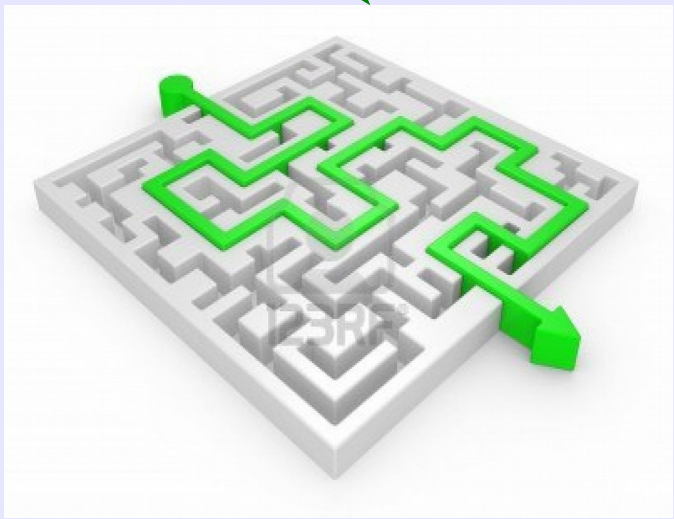
Introducción/Contextualización

Motivación

AND



OR

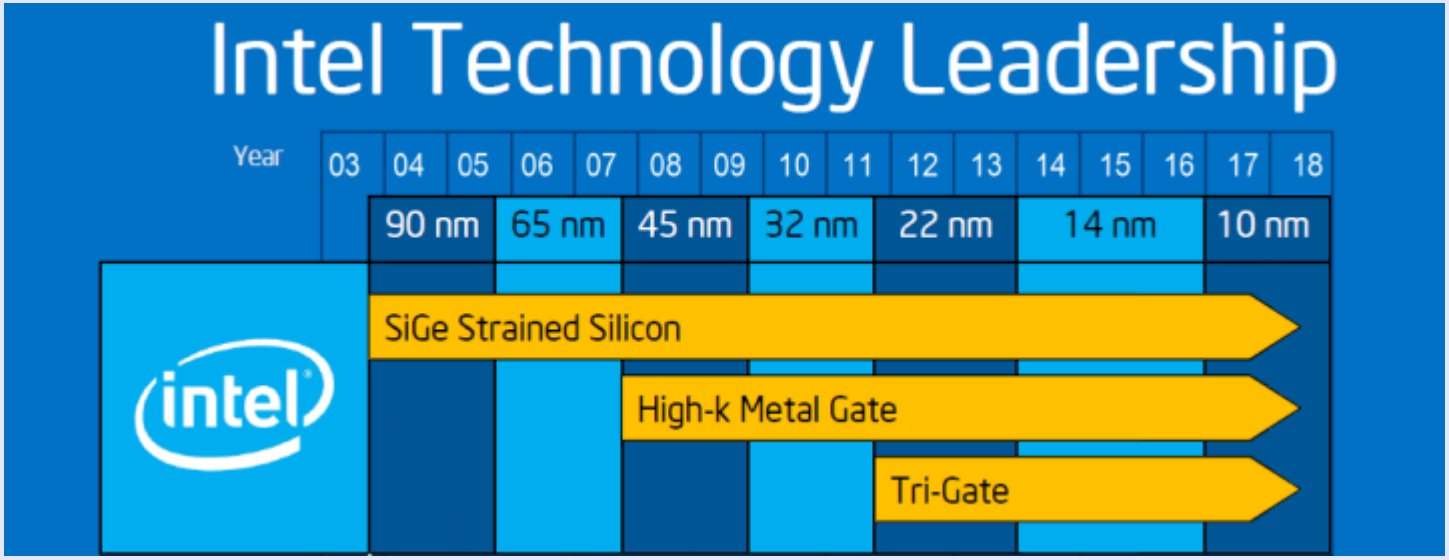


Motivación

- Los sistemas de hardware y software son cada vez más grandes y complejos.
  - El número de transistores el año 1979 en procesador 8088 era de 29,000 a 3 micrones.
  - En Pentium-4 del año 2000 era de 42 millones a 0.18 micrones.
  - En procesador Dual-core del año 2005 es de 290 millones a 0.065 micrones.
  - En Core i7 del año 2011 es de 1170 millones a 0.022 micrones
- Además, dependemos de nuestros sistemas computacionales más que nunca.

Motivación

- Los sistemas de hardware y software son cada vez más grandes y complejos.



IBM ya está en los 5nm: 30.000 millones de transistores en el tamaño de una uña

Motivación

- Cada vez es más fácil que se produzcan errores en las aplicaciones
  - Son más difíciles de detectar
- Problema
  - En ciertas aplicaciones no pueden tolerarse errores
    - Diagnóstico y aparatos médicos
    - Misiones espaciales
- Mientras antes se detecten los errores, menos costo
  - A Intel, el problema del punto-flotante en procesadores Pentium les costo 500 millones de dólares.

Motivación

- Implementación de un programa (software)
  - Comprobar
    - Satisface especificación
    - Tiene la funcionalidad esperada
- ¿Quién se encarga de esto?
  - Validación
    - ¿Estamos construyendo el producto correcto?
  - Verificación
    - ¿Estamos construyendo el producto correctamente?
- ¿Cuándo?
  - En cada etapa del proceso de desarrollo del software

Motivación

- Al finalizar cada fase
  - Comprobar que el trabajo realizado hasta el momento cumple con los objetivos previstos
  - Decidir si está preparado o no para pasar a la siguiente fase

Introducción

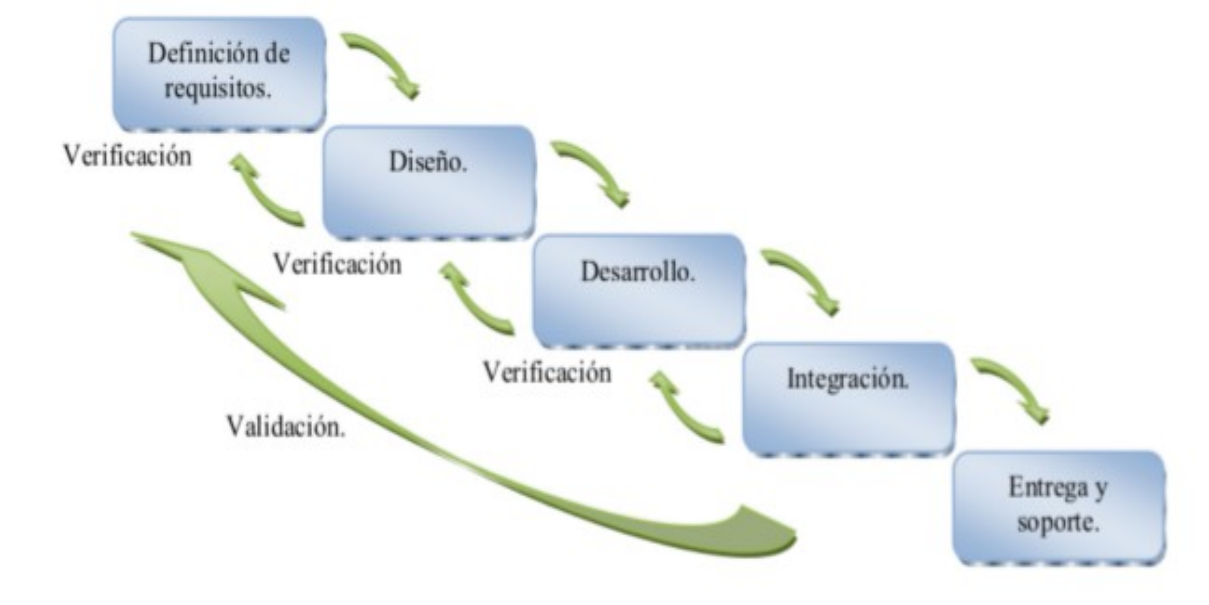
- Verificación del software
  - Comprobar que el software está de acuerdo con su especificación
  - Cumple con los requerimientos especificados
    - Requerimientos funcionales y no funcionales
  - Todas las actividades que son llevadas a cabo para averiguar si el software cumple con sus objetivo
  - *The process of evaluating a system or component to determine whether the product of a given development phase satisfay the conditions imposed at the start of the phase*.
    - Definición de IEEE

Introducción

- Validación de software
  - Busca comprobar que el software hace lo que el usuario espera.
  - El proceso asegura que el software fabricado se comporta como se espera y de acuerdo a las expectativas del cliente.
  - ¿El software cumple las espectativas del cliente?
  - *The process of evaluating a system or component during or at the end of the development process to determine whether it satisfiesspecified requirements*

Introducción

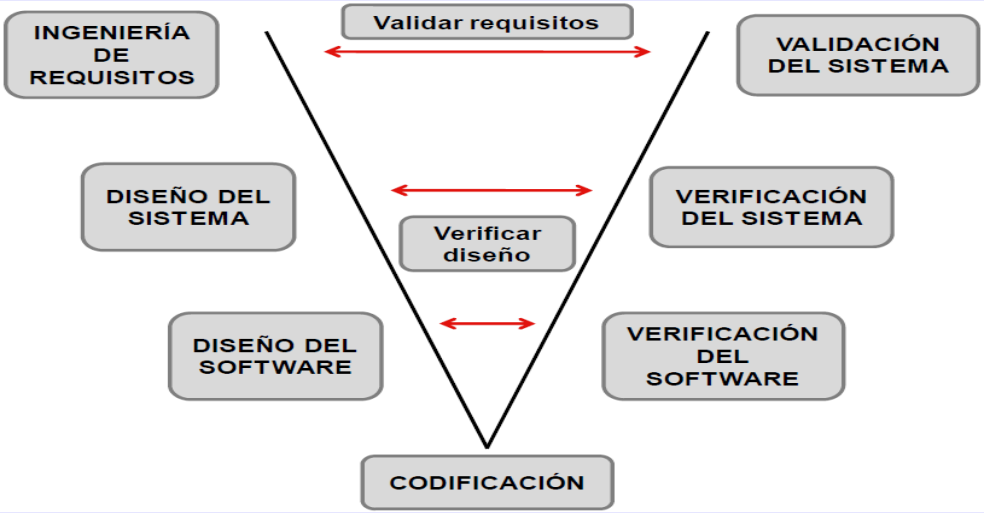
- Proceso de validación y verificación (*V&V*)
  - Uso de procedimientos, herramientas, actividades, técnicas a la par que se desarrolla el software
    - Software resuelve el problema planteado



Introducción

- V&V en el análisis y especificación de requisitos
  - Comprobar que el proyecto es viable
  - Comprobar que las especificaciones documentadas son completas, correctas, precisas, evaluables
    - Responden a las expectativas de los clientes
- V&V del diseño
  - Los requisitos no son incompletos
  - Los requisitos no están incorrectamente diseñados
- V&V en la implementación y la codificación
  - Pruebas de software

- V&V operación y mantenimiento del software
  - Al realizar un cambio
    - Examinar el impacto de éste sobre el sistema
    - ¿Qué actividades repetir?
      - Garantizar misma calidad que antes del cambio



Introducción

- Técnicas para el proceso de validación & verificación
  - Técnicas estáticas (analíticas-inspección de código)
    - Analizan el producto para deducir su correcta operación
    - No implican ejecución de código
  - Técnicas dinámicas (pruebas)
    - Experimentar con el comportamiento de un producto para ver si el producto actúa como es esperado
    - Implican ejecución de código
  - Ejecución simbólica
    - Técnica híbrida.

Introducción

- Ejemplo:
  - Programa que lee tres números enteros (lados de un triángulo). El resultado es un mensaje indicando el tipo de triángulo: isósceles, escaleno, equilátero
- Verificación estática
  - Revisamos el código para detectar defectos

```
if l1=l2 or l2=l3 then
   write (“equilatero”)
else
   …
```

  - Se puede revisar solo o por grupos

Introducción

- Técnicas dinámica
  - Probamos el código con diferentes casos de prueba
    - lado1=2, lado2=2, lado3=2 → equilátero
    - lado1=2, lado2=2, lado3=3 → isósceles
  - Comparamos el resultado obtenido obtenido con el esperado
    - Si no coinciden habrá un fallo.

Técnicas dinámicas
Pruebas

- Principal técnica de verificación y validación
  - Ejecutar el programa con datos similares a los reales
  - Se examinan las salidas y se buscan anomalías
  - Necesitamos una versión operativa del producto
- Diseñadas para revelar la presencia de defectos
  - Una prueba es exitosa si pone en evidencia al producto
- Única técnica de validación para los requerimientos no funcionales
  - El software debe ser ejecutado para ver su comportamiento

Técnicas dinámicas
Pruebas

- Dos tipos de pruebas
  - Pruebas de validación
    - El software es el que el cliente quiere
      - Cumple sus requerimientos
    - Tiene éxito si demuestra que un requisito se ha implementado correctamente
  - Pruebas de defectos
    - Revelar defectos en el sistema
    - Inconsistencia entre un programa y su especificación
    - Demuestran la presencia, y no la ausencia, de fallos

Técnicas dinámicas Pruebas

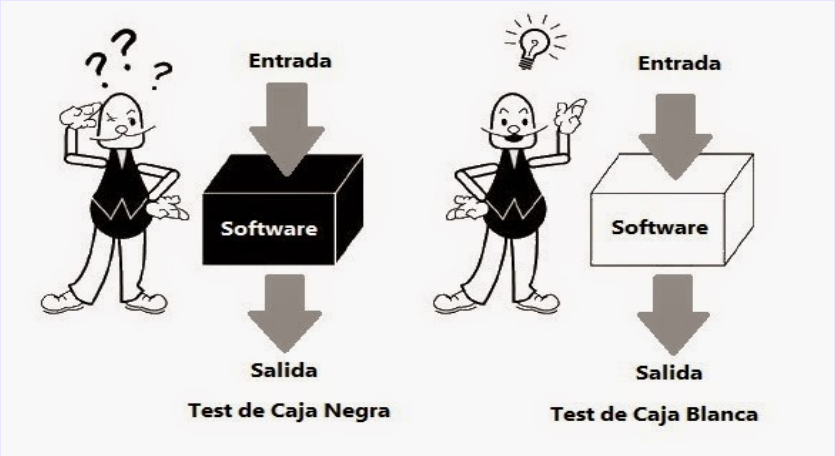
- Pruebas de validación
  - Pruebas de aceptación
    - Desarrolladas por el cliente
  - Pruebas alfa
    - Realizadas por el usuario en un entorno controlado
      - Desarrollador como observador
  - Pruebas beta
    - Realizadas por el usuario en su entorno de trabajo
  - Pruebas estadísticas
    - Probar el rendimiento y la fiabilidad
    - Se diseñan para reflejar la carga de trabajo habitual
    - Número de caída y tiempos de respuesta

Técnicas dinámicas

- Pruebas por defectos
  - Detectar defectos latentes de un sistema software antes de entregar el producto
    - Es exitosa si descubre un comportamiento contrario a la especificación → Error
  - No son posibles pruebas exhaustivas
    - Subconjuntos de casos de prueba → políticas para establecerlos
      - Que todas las instrucciones se ejecuten al menos una vez
      - Probar todas las funciones que se acceden a través de un menú
      - Probar con entrada correcta e incorrectas

Técnicas dinámicas

- Pruebas por defectos
  - Pruebas de caja negra o funcionales
    - Las pruebas se seleccionan en función de la especificación y no de la estructura interna del software
  - Pruebas de caja blanca o estructurales
    - Las pruebas se seleccionan en función del conocimiento que se tiene de la implementación



Técnicas estáticas

- Características
  - Correspondencia entre un programa y su especificación
  - No se puede demostrar que el software es útil
  - No sirven para comprobar ciertas propiedades
    - Rendimiento y fiabilidad
- Tipos
  - Análisis de código (Inspecciones)
  - Análisis automatizado de código fuente
  - Verificación formal

Técnicas estáticas

- Análisis de código (Inspecciones)
  - Se revisa el código buscando defectos
    - Cualquier representación legible del software
      - Requerimientos o modelo de diseño
  - Se puede llevar a cabo en grupo
    - Diferentes roles
  - Muy utilizado en ingeniería de sistemas críticos
  - Listas de comprobación
    - Dependen del lenguaje, estándares, prácticas locales
    - Hay que actualizarlas

Técnicas estáticas
Inspecciones

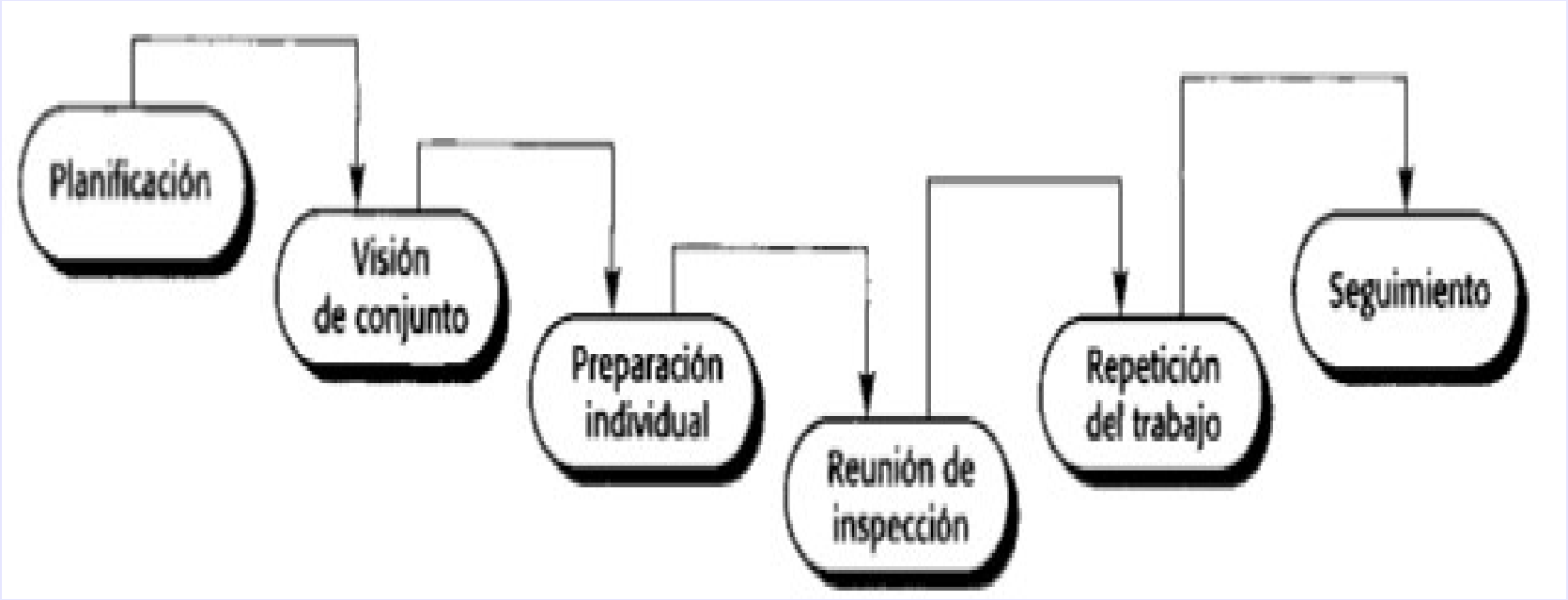
- Roles del proceso de inspección (Michael Fagan, 1970)
  - Autor o propietario
    - El diseñador o programador responsable del documento
    - Responsable de reparar los defectos descubiertos
  - Inspector
    - Persona que examina el producto
    - Encuentra errores, omisiones o inconsistencias
  - Lector
    - Presenta el documento en la reunión de inspección
  - Secretario
    - Registra los resultados de la reunión de inspección

Técnicas estáticas
Inspecciones

- Roles del proceso de inspección (Michael Fagan, 1970)
  - Presidente o moderador
    - Gestiona el proceso y facilita la inspección
    - Realiza un informe de los resultados del proceso para el moderador jefe
  - Moderador jefe
    - Responsable de las mejoras del proceso de inspección, actualización de las listas de comprobación, estándares de desarrollo, etc.

Técnicas estáticas
Inspecciones

- Proceso de inspección (Michael Fagan, 1970)



Técnicas estáticas
Inspecciones

- Proceso de inspección (Michael Fagan, 1970)
  - Planificación
    - El moderador planea la inspección
  - Visión de conjunto
    - El autor describe los antecedentes del producto
  - Preparación individual
    - Cada inspector examina el producto para identificar posibles defecto
  - Reunión de inspección
    - El lector lee parte por parte del producto y los inspectores indican los defectos de cada parte.

Técnicas estáticas
Inspecciones

- Proceso de inspección (Michael Fagan, 1970)
  - Repetición del trabajo
    - El autor realiza cambios en el producto de acuerdo a los planes de acción de la reunión de la inspección
  - Seguimiento
    - Los cambios del autor son revisados para asegurarse de que todo está correcto

Técnicas estáticas
Inspecciones

- Listas de comprobaciones
  - Listas de comprobaciones de libros
  - Conocimiento de los defectos que son comunes en un dominio de aplicación particular

Defectos de datos	¿Se inicializan todas las variables antes de que se utilicen sus valores? ¿Tienen nombre todas las constantes? ¿El límite superior de los vectores es igual al tamaño del vector o al tamaño 21? Si se utilizan cadenas de caracteres, ¿tienen un delimitador explícitamente asignado? ¿Existe alguna posibilidad de que el búfer se desborde?
Defectos de control	Para cada sentencia condicional, ¿es correcta la condición? ¿Se garantiza que termina cada bucle? ¿Están puestas correctamente entre llaves las sentencias compuestas? En las sentencias case, ¿se tienen en cuenta todos los posibles casos? Si se requiere una sentencia break después de cada caso en las sentencias case, ¿se ha incluido?
Defectos de entrada/salida	¿Se utilizan todas las variables de entrada? ¿Se les asigna un valor a todas las variables de salida? ¿Pueden provocar corrupciones de datos las entradas no esperadas?
Defectos de interfaz	¿Las llamadas a funciones y a métodos tienen el número correcto de parámetros? ¿Concuerdan los tipos de parámetros reales y formales? ¿Están en el orden correcto los parámetros? Si los componentes acceden a memoria compartida, ¿tienen el mismo modelo de estructura de la memoria compartida?
Defectos de gestión de almacenamiento	Si una estructura enlazada se modifica, ¿se reasignan correctamente todos los enlaces? Si se utiliza almacenamiento dinámico, ¿se asigna correctamente el espacio de memoria? ¿Se desasigna explícitamente el espacio de memoria cuando ya no se necesita?
Defectos de manejo de excepciones	¿Se tienen en cuenta todas las condiciones de error posibles?

Técnicas estáticas
Inspecciones

- Listas de comprobaciones
  - Especificación de requisitos
    - Requisitos o especificaciones
      - Descripción mala de los requisitos de los usuarios.
    - Funcionalidad
      - Características del sistema incorrectas o incompatibles.
    - Interfaz de Usuario, Software y Hardware:
      - Especificaciones incorrectas sobre como el producto interacciona con el entorno.
    - Descripción funcional:
      - Descripciones mal hechas sobre que hace el producto.

Técnicas estáticas
Inspecciones

- Listas de comprobaciones
  - Diseño
    - Hardware, software e interfaz de usuario
      - Errores con el producto sobre la interacción del entorno con el usuario.
    - Descripción Funcional:
      - El diseño no hace lo que el módulo o producto debiera. Son defectos encontrados durante la inspección del diseño o durante la implementación.
    - Comunicaciones entre procesos
      - No existe comunicación entre interfaz o módulos del sistema en desarrollo y se pierden los datos.
    - Definición de datos:
      - Diseño incorrecto de las estructuras de datos que se utilizarán en los módulos del sistema en desarrollo.

Técnicas estáticas
Inspecciones

- Listas de comprobaciones
  - Diseño
    - Diseño del módulo
      - Problemas con el flujo de control y ejecución entre procesos.
    - Descripción de la lógica
      - El diseño es incorrecto en la lógica comparado con el análisis. Es un defecto encontrado durante la inspección o la implementación.
    - Chequeo de errores
      - Incorrecta verificación de las condiciones de error dentro del sistema en desarrollo.
    - Estándares
      - El diseño no cumple con los estándares internos.

Técnicas estáticas
Inspecciones

- Listas de comprobaciones
  - Código

Defectos de datos	¿Se inicializan todas las variables antes de que se utilicen sus valores? ¿Tienen nombre todas las constantes? ¿El límite superior de los vectores es igual al tamaño del vector o al tamaño 21? Si se utilizan cadenas de caracteres, ¿tienen un delimitador explícitamente asignado? ¿Existe alguna posibilidad de que el búfer se desborde?
Defectos de control	Para cada sentencia condicional, ¿es correcta la condición? ¿Se garantiza que termina cada bucle? ¿Están puestas correctamente entre llaves las sentencias compuestas? En las sentencias case, ¿se tienen en cuenta todos los posibles casos? Si se requiere una sentencia break después de cada caso en las sentencias case, ¿se ha incluido?
Defectos de entrada/salida	¿Se utilizan todas las variables de entrada? ¿Se les asigna un valor a todas las variables de salida? ¿Pueden provocar corrupciones de datos las entradas no esperadas?
Defectos de interfaz	¿Las llamadas a funciones y a métodos tienen el número correcto de parámetros? ¿Concuerdan los tipos de parámetros reales y formales? ¿Están en el orden correcto los parámetros? Si los componentes acceden a memoria compartida, ¿tienen el mismo modelo de estructura de la memoria compartida?
Defectos de gestión de almacenamiento	Si una estructura enlazada se modifica, ¿se reasignan correctamente todos los enlaces? Si se utiliza almacenamiento dinámico, ¿se asigna correctamente el espacio de memoria? ¿Se desasigna explícitamente el espacio de memoria cuando ya no se necesita?
Defectos de manejo de excepciones	¿Se tienen en cuenta todas las condiciones de error posibles?

Técnicas estáticas
Inspecciones

- Listas de comprobaciones
  - Documentación
    - Errores en manuales, instrucciones de instalación, demostraciones, todos ello centrado al cliente.
  - Entorno de apoyo
    - Software de pruebas
      - Problemas de software utilizado para probar el producto
    - Hardware de pruebas
      - Problemas del hardware de pruebas
    - Herramienta de desarrollo
      - Si no se comportan adecuadamente
    - Software de integración:
      - Problemas de interacción software/herramientas.

Técnicas estáticas
Inspecciones

- Ventajas
  - No existe interacción entre errores
    - Una única sesión → muchos errores
  - Pueden inspeccionarse versiones incompletas
    - No aumenta el coste
  - Pueden considerarse atributos de calidad
    - Cumplimiento de estándares
    - Portabilidad
    - Mantenimiento

Técnicas estáticas
Inspecciones

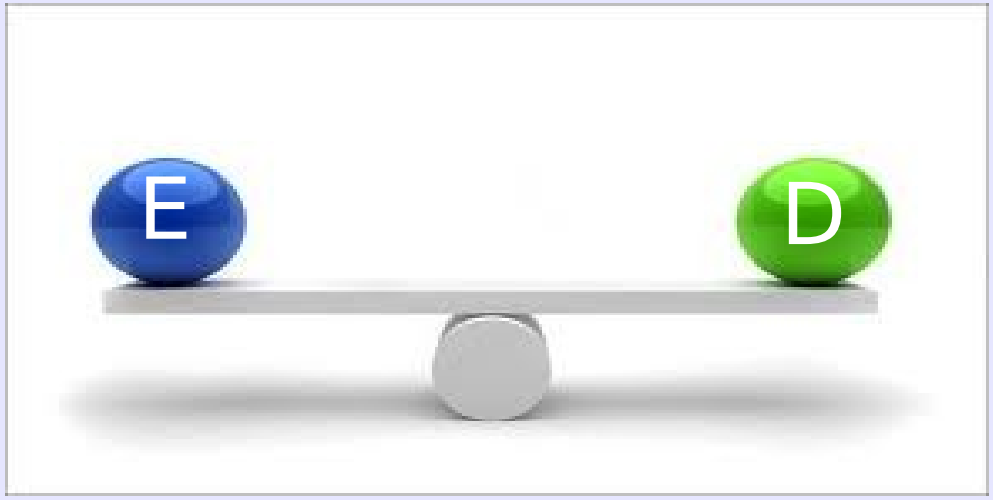
- Eficacia → Varios estudios
  - + 60 % errores (*Fangan, 1986*)
    - Inspecciones informales
  - 90% (*Mill et al, 1987*)
    - Inspecciones más formales (corrección de argumentos)
  - + Efectiva y - costosa que las pruebas (*Selby et al, 1987*) y (*Gilb y Graham, 1993*)
    - Defectos del programa
- Inspecciones en vez de pruebas de componentes

Técnicas estáticas
Inspecciones

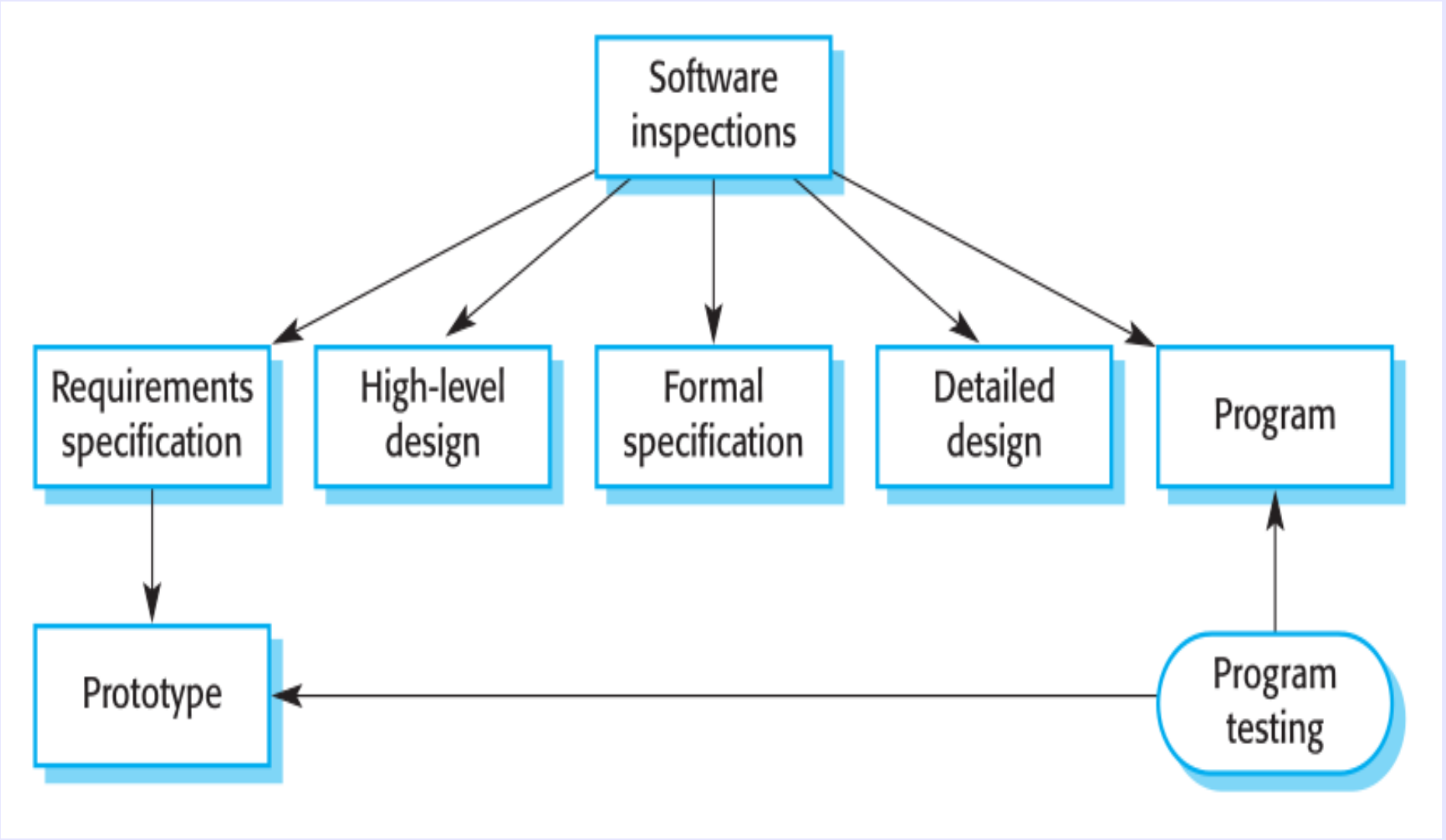
- Reticencias de uso en las empresas
  - Las inspecciones no son más efectivas que las pruebas
    - Ingenieros software expertos en pruebas
  - Costes adicionales durante el diseño y el desarrollo
    - Gestores
  - No hay ahorro de tiempo en la depuración
    - Gestores

Técnicas estáticas
Inspecciones

- Equilibrio entre estática y dinámica
  - Primeras etapas desarrollo → Inspecciones
  - Integración sistema → pruebas
  - Ejemplo → Revisión de los casos de prueba
    - Encontrar problemas en estos test
    - Diseñar formas más efectivas de probar el sistema



Técnicas estáticas



[1] Ian Sommerville, “Software Engineering 7”, Addison-Wesley, 2004

Análisis Estático Automático

- Análisis estático automatizado
  - Posibilidad de automatizar las comprobaciones de los programas frente a las listas
    - Ciertos errores y heurísticas
  - Creación de analizadores estáticos
    - Llamar la atención sobre las anomalías del programa
      - Variables que no se usan
      - Variables si inicializar
      - Datos cuyo valor es inalcanzable
    - Son resultado de errores de programación u omisiones
    - Muy útiles en lenguajes que no tienen reglas estrictas
      - Desarrollo de sistemas críticos

- Comprobaciones analizadores estáticos

Clase de defecto	Comprobación del análisis estático
Defectos de datos	Variables utilizadas antes de su inicialización. Variables declaradas pero nunca utilizadas. Variables asignadas dos veces pero nunca utilizadas entre asignaciones. Posibles violaciones de los límites de los vectores. Variables no declaradas.
Defectos de control	Código no alcanzable. Saltos incondicionales en bucles.
Defectos de entrada/salida	Las variables salen dos veces sin intervenir ninguna asignación.
Defectos de interfaz	Inconsistencias en el tipo de parámetros. Inconsistencias en el número de parámetros. Los resultados de las funciones no se utilizan. Existen funciones y procedimientos a los que no se les llama.
Defectos de gestión de almacenamiento	Punteros sin asignar. Aritmética de punteros.

Análisis Estático Automático

- Etapas del análisis estático
  - Análisis de flujo de control
    - Bucles con múltiples salidas o entradas
    - Código no alcanzable
  - Análisis del uso de datos
    - Variables no inicializadas, o no usadas
  - Análisis de interfaces
    - Declaración de funciones y su utilización
  - Análisis de flujo de información
    - Dependencia entre las variables de entrada y salida
  - Análisis de caminos
    - Desenreda el control del programa

Análisis Estático Automático

- Problemas
  - No pueden sustituir totalmente a las inspecciones
  - Existen errores que no pueden detectar
    - Inicializaciones incorrectas
    - Argumentos incorrectos de tipo correcto

- Mejoras
  - Inclusión de anotaciones
    - Definición de restricciones y comentarios con estilo de programa

Técnicas estáticas
Verificación Formal

- Verificación formal
  - Estudia los fundamentos teóricos y la implementación de técnicas de verificación de sistemas computacionales
    - Técnicas matemáticas para asegurar la validez de un sistema respecto a una especificación
    - Éxito de aplicación del razonamiento automático a la computación
    - Son cada vez más aplicadas a la industria
      - IBM, AT&T, Motorola, ...

Técnicas estáticas
Verificación Formal

- Un programa es correcto si cumple con la especificación
- Objetivo
  - Demostrar que un programa es correcto a partir de una especificación formal
    - Análisis detallado de la especificación del sistema y del programa
    - Representaciones matemáticas del software
- Uso principal
  - Orientado al desarrollo de sistemas críticos y seguros
    - Consume tiempo y es caro

Técnicas estáticas
Verificación Formal

- Tareas principales
  - Modelado
    - Se construye un modelo matemático de los posibles comportamientos del sistema
  - Especificación
    - Se especifica en un lenguaje formal los comportamientos deseables del sistema
  - Verificación
    - Se chequea si el modelo satisface la especificación
    - Verificar formalmente que una representación del sistema es semánticamente equivalente a otra

Técnicas estáticas
Verificación Formal

- Formal
  - El modelo y la especificación son objetos matemáticos
- Verificación
  - El análisis responde con certeza si la especificación se cumple o no

Ordenar : vector A × int p × int q → vector B

function Ordenar(vector A, int p, int q):vector B {

Precondición

$\{0 \le p \le q+1\}$

S1

$$\left\{\begin{array}{l}(\forall i : s \le i < q : \forall j : p \le j < s : val(B,j) \le val(B,i)) \wedge \\ p \le s \le q \wedge EsPermutacion(A,B,p,q)\end{array}\right\}$$

S2

Postcondicion

$\{(\forall i : p \le j \le i < q : val(B,j) \le val(B,i)) \wedge EsPermutacion(A,B,p,q)\}$

}

Técnicas estáticas
Verificación Formal

- Verificación deductiva
  - Probar un teorema desde un conjunto de axiomas
    - Método más general
    - Demostrador automático de teoremas
- Verificación automatizada (model checking)
  - La verificación de la especificación en el modelo se realiza de forma algorítmica
    - Simular todas los posibles comportamientos en busca de errores

Verificación Formal

- Ventajas
  - Fuerza un análisis detallado de la especificación
  - Revelar inconsistencias u omisiones potenciales
    - No desveladas hasta que el software es operacional
  - Los errores de implementación no comprometen su confiabilidad
    - El programa satisface su especificación

Verificación Formal

- Objeciones
  - Demostración más larga (y compleja) que el propio programa
  - La demostración puede ser incorrecta
  - Demasiada matemática para el programador medio
  - No se consideran limitaciones del hardware
  - La especificación puede ser incorrecta
    - Nada es seguro 100%
  - Demasiado caro

Ejecución simbólica

- Definición
  - Sustituir variables de entrada por valores simbólicos y de forma estática ir evaluando los posibles dominios para las variables a lo largo de una trayectoria del flujo del control.
- Objetivo
  - Identificar restricciones entre los valores simbólicos de la entrada, y bajo los cuales se ejecutó la trayectoria seleccionada

Ejecución simbólica

- Ejemplo

[x = X, a = A, y = Y]

x := y + 2;

[x = Y + 2, a = A, y = Y]

if x > a then

a := a + 2; [x=Y+2, a=A+2,y=Y] [Y+2>A]

else

y := x + 3;

end if

x := x + a + y;

[x = 2Y +2+A , a=A+2, y=Y] [Y + 2 > A]

Ejecución simbólica

- Ejemplo

[x = X, a = A, y = Y]

x := y + 2;

[x = Y + 2, a = A, y = Y]

if x > a then

    a := a + 2;

else

    y := x + 3; [x = Y + 2, a = A, y = Y + 5] [Y + 2 <= A]

end if

x := x + a + y;

[x = 2Y + A + 7, a = A, y = Y + 5] [Y + 2 <= A]

Ejecución simbólica

```
y = read()
y = 2 * y
if (y == 12)
    fail()
print("OK")
```

- ¿Qué tiene que ocurrir para que se ejecute *fail*?
  - Se le asigna un valor 's' a la entrada y se trabaja con él
  - 2*s ==12 → restricción para que se ejecute *fail*