



## Unidad 7 – Pruebas Implementación y Mantenimiento

Un **error** de programación se puede dar al momento en que el desarrollador asigna 2 valores a una misma variable, o cometido en la lógica de programación. En el momento en que se compila el código, se arma la versión y se instala en un ambiente, ese software contiene **defectos**, ¿Cuáles?, no lo sabemos si no hasta ejecutar nuestras pruebas, en el momento en que el sistema falla, se manifiesta mediante un mensaje de error el cual capturamos para reportar un **fallo**.

```
void MinMax (int Min, int Max)
{
    int Help;
    if (Min>Max)
    {
        Max = Help;
        Max = Min;
        Help = Min;
    }
}
End MinMax;
```

## Error (“Error”):

- Acción humana que produce un resultado incorrecto.
- Ejemplo: Un error de programación

## Defecto (“Defect”):

- Desperfecto en un componente o sistema que puede causar que el componente o sistema falle en desempeñar las funciones requeridas.
- **Ejemplo: Una sentencia o una definición de datos incorrectas.**





## Fallo ("Failure"):

- Manifestación física o funcional de un defecto.
- **Ejemplo: Desviación de un componente o sistema respecto de la prestación, servicio o resultado esperados.**


En resumen, un **error** introduce un **defecto** en el software que a su vez causa un **fallo** al momento de ejecutar pruebas.

Es común ver como algunas personas utilizan de manera indistinta los términos Defecto, Falla y Error. Sin embargo, cada uno de ellos tiene un significado diferente:


**Defecto:** Un defecto se encuentra en un artefacto y puede definirse como una diferencia entre la versión correcta del artefacto y una versión incorrecta. Coincide con la definición de diccionario, "**imperfección**".

**Falla:** En terminología IEEE, una falla es la **discrepancia visible** que se produce al ejecutar un programa con un defecto, el cual es incapaz de funcionar correctamente (no sigue su curso normal).

**Error:** Es una equivocación cometida por el desarrollador. Algunos ejemplos de errores son: un error de digitación, una malinterpretación de un requerimiento o de la funcionalidad de un método. El estándar 829 de la IEEE coincide con la definición de diccionario de error como "**una idea falsa o equivocada**". Por tal razón un programa no puede tener o estar en un error, ya que los programas no tienen ideas; las ideas las tienen la gente.



Ampliando el tema de los errores, estos errores ocurren cuando cualquier aspecto de un producto de software es incompleto, inconsistente o incorrecto. Las tres grandes clases clasificaciones de errores del software son los de **requisitos**, de **diseño** y de **implementación**. En esta sección hablaremos de dos de ellos:







## Errores de diseño:

Se introducen por fallas al traducir los requisitos en estructuras de solución correctas y completas, por inconsistencias tanto dentro de las especificaciones de diseño y como entre las especificaciones de diseño y los requisitos.

Un error de requisitos o un error de diseño, que no se descubre sino hasta las pruebas de código fuente, **puede ser muy costoso de corregir**. De modo que es importante que la calidad de los requisitos y de los documentos del diseño se valoren pronto y con frecuencia.

## Los errores de instrumentación:

Son los cometidos al traducir las especificaciones de diseño en **código fuente**. Estos errores pueden ocurrir en las declaraciones de datos, en las referencias a los datos, en la lógica del flujo de control, en expresiones computacionales, en interfaces entre subprogramas y en operaciones de entrada/salida.



## Los errores de instrumentación:

La calidad de los productos de trabajo generados durante el análisis y el diseño se puede estimar y mejorar utilizando procedimientos sistemáticos de control de calidad, mediante **recorridos e inspecciones y por medio de verificaciones automatizadas** para supervisar que sea consistente y que esté completo.



## La prueba

Es un conjunto de actividades que pueden planearse por adelantado y realizarse de manera sistemática. Por esta razón, durante el proceso de software, debe definirse una plantilla para la prueba del software: un conjunto de pasos que incluyen métodos de prueba y técnicas de diseño de casos de prueba específicos.

## 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.

Boehm [Boe81] afirma esto de esta forma:

Verificación: “¿Construimos el producto correctamente?”

Validación: “¿Construimos el producto correcto?”

## Estrategia de prueba



## Estrategia de prueba

La **prueba de unidad** comienza en el vértice de la espiral y se concentra en cada unidad (por ejemplo, componente, clase o un objeto de contenido de una *webapp*) del software como se implementó en el código fuente.

La prueba avanza al moverse hacia afuera a lo largo de la espiral, hacia la **prueba de integración**, donde el enfoque se centra en el diseño y la construcción de la arquitectura del software.

Al dar otra vuelta hacia afuera de la espiral, se encuentra la **prueba de validación**, donde los requerimientos establecidos como parte de su modelado se validan confrontándose con el software que se construyó.

Finalmente, se llega a la **prueba del sistema**, donde el software y otros elementos del sistema se prueban como un todo.

## Estrategia de prueba

Inicialmente, las pruebas se enfocan en cada componente de manera individual, lo que garantiza que funcionan adecuadamente como unidad. De ahí el nombre de ***prueba de unidad***. Esta prueba utiliza mucho de las técnicas de prueba que ejercitan rutas específicas en una estructura de control de componentes para asegurar una cobertura completa y la máxima detección de errores.



## Estrategia de prueba

A continuación, los componentes deben ensamblarse o integrarse para formar el paquete de software completo. La *prueba de integración* aborda los conflictos asociados con los problemas duales de verificación y construcción de programas. Durante la integración, se usan más las técnicas de diseño de casos de prueba que se enfocan en entradas y salidas, aunque también pueden usarse técnicas que ejercitan rutas de programa específicas para asegurar la cobertura de las principales rutas de control.

## Estrategia de prueba

Después de integrar (construir) el software, se realiza una serie de pruebas de orden superior.

Deben evaluarse criterios de validación (establecidos durante el análisis de requerimientos).

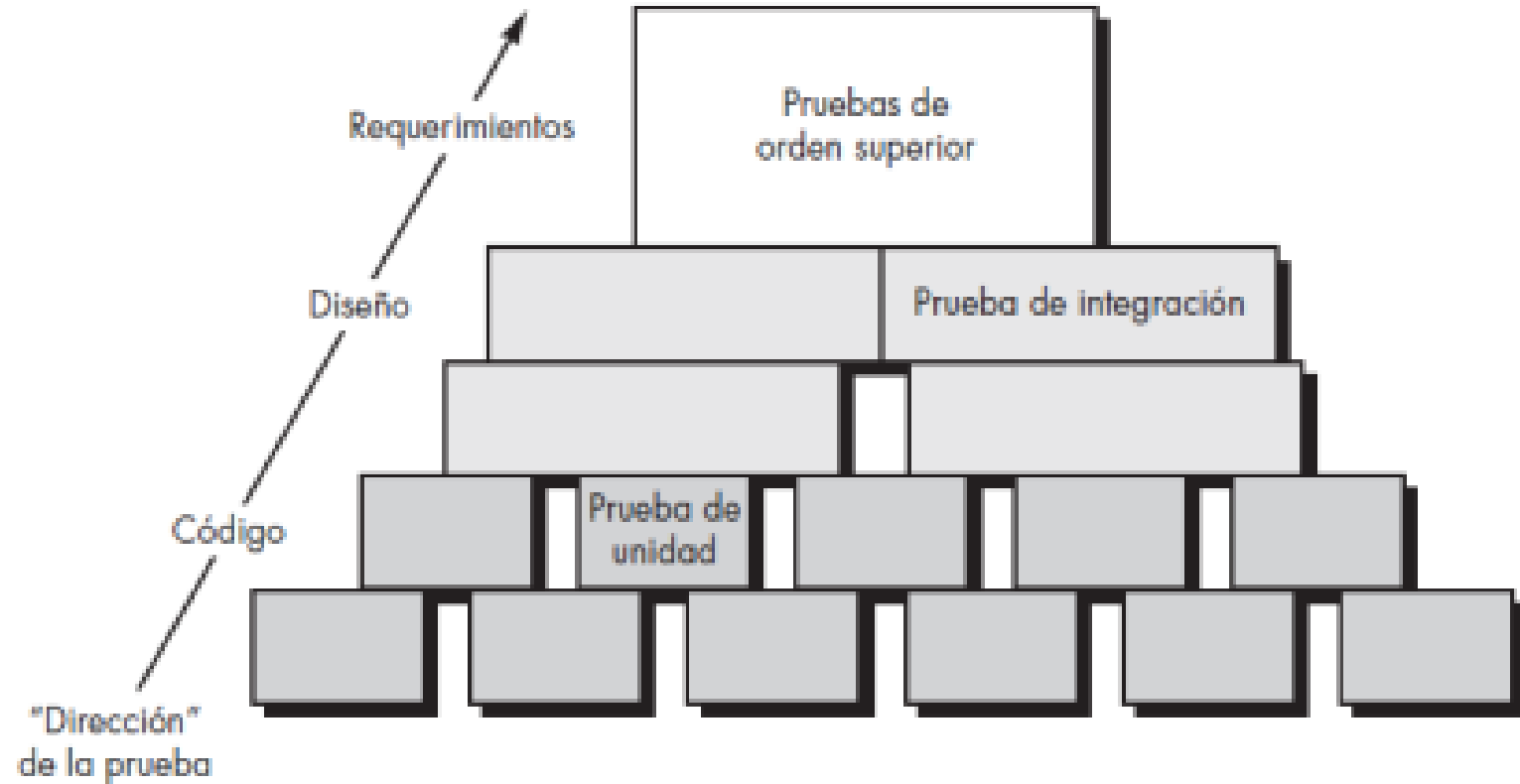
La prueba de validación proporciona la garantía final de que el software cumple con todos los requerimientos informativos, funcionales, de comportamiento y de rendimiento.


El último paso de la prueba de orden superior. La prueba del sistema verifica que todos los elementos se mezclan de manera adecuada y que se logra el funcionamiento/rendimiento global del sistema.

# Estrategia de prueba


FIGURA 17.2

Pasos de la prueba del software






La *prueba alfa* se lleva a cabo en el sitio del desarrollador por un grupo representativo de usuarios finales. El software se usa en un escenario natural con el desarrollador “mirando sobre el hombro” de los usuarios y registrando los errores y problemas de uso. Las pruebas alfa se realizan en un ambiente controlado.



La *prueba beta* se realiza en uno o más sitios del usuario final. A diferencia de la prueba alfa, por lo general el desarrollador no está presente. Por tanto, la prueba beta es una aplicación “en vivo” del software en un ambiente que no puede controlar el desarrollador. El cliente registra todos los problemas (reales o imaginarios) que se encuentran durante la prueba beta y los reporta al desarrollador periódicamente. Como resultado de los problemas reportados durante las pruebas beta, es posible hacer modificaciones y luego preparar la liberación del producto de software a toda la base de clientes.



La *prueba de caja negra* se refiere a las pruebas que se llevan a cabo en la **interfaz del software**. Una prueba de caja negra examina algunos aspectos fundamentales de un sistema con **poca preocupación** por la estructura **lógica interna** del software.

La *prueba de caja blanca* del software se basa en el examen cercano de los **detalles de procedimiento**. Las rutas lógicas a través del software y las colaboraciones entre componentes se ponen a prueba al revisar conjuntos específicos de condiciones y/o bucles.



## Herramientas de Software

Auxiliar al equipo de software en el desarrollo de un conjunto completo de casos de prueba tanto para prueba de caja negra como de caja blanca.

-Herramientas de prueba estática

**1- Basadas en código:** aceptan código fuente como entrada y realizan algunos análisis que dan como resultado la generación de casos de prueba

**2- Lenguaje de pruebas especializado:** permiten al ingeniero de software escribir especificaciones de prueba detalladas que describen cada caso de prueba y la logística para su ejecución

**3- Basadas en requerimientos:** aíslan requerimientos de usuario específicos y sugieren casos de prueba (o clases de pruebas) que revisarán los requerimientos.

-Herramientas de prueba dinámica: interactúan con un programa en ejecución.



## El mantenimiento

El mantenimiento del software es también una de las fases del ciclo de vida del desarrollo del sistema (SDLC), que se aplica al desarrollo de software. La fase de mantenimiento es la fase que sigue al despliegue (implementación) del software en el campo.



## El mantenimiento

El mantenimiento de software es una actividad muy amplia que incluye corrección de errores, mejoras de capacidad, eliminación de funciones obsoletas y optimización. Debido a que el cambio es inevitable, se deben desarrollar mecanismos de evaluación, control y modificación.



## El mantenimiento

Análisis y diseño conducen a una importante característica del software que se llamará **mantenibilidad**. En esencia, la mantenibilidad es un indicio cualitativo de la facilidad con la que el software existente puede corregirse, adaptarse o aumentarse.

Mientras más difícil sea entender un programa, más difícil será darle mantenimiento.



## El mantenimiento

- **Mantenimiento preventivo:** Consiste en la revisión constante del software para detectar posibles fuentes de problemas que puedan surgir en el futuro.



## El mantenimiento

- Mantenimiento predictivo. Evalúa el flujo de ejecución del programa para predecir con certeza cuándo ocurrirá la falla, y así determinar cuándo es apropiado hacer los ajustes correspondientes.





## El mantenimiento

- Mantenimiento correctivo. Corrige los defectos encontrados en el software, y que originan un comportamiento diferente al deseado. Estas fallas pueden ser de procesamiento, rendimiento (por ejemplo, uso ineficiente de recursos de hardware), programación (inconsistencias en la ejecución), seguridad o estabilidad, entre otras.



## El mantenimiento

- Mantenimiento adaptativo. Si es necesario cambiar el entorno en el que se utiliza la aplicación (que incluye el sistema operativo, la plataforma de hardware o, en el caso de las aplicaciones web, el navegador), puede ser necesario modificarla para mantener su plena funcionalidad en estas nuevas condiciones.



## El mantenimiento

- **Mantenimiento evolutivo.** Es un caso especial donde la adaptación es prácticamente obligatoria, ya que de lo contrario el programa quedaría obsoleto con el paso del tiempo. Por ejemplo, el cambio de versión en un navegador (a menudo impuesto sin el consentimiento del usuario) suele requerir ajustes en los plugins y aplicaciones web.



## El mantenimiento

- Mantenimiento perfecto. Por diferentes razones, el usuario puede solicitar la adición de nuevas funcionalidades o características no consideradas en el momento de la implementación del software. Un mantenimiento perfecto adapta la aplicación a este requisito. El mantenimiento permanente del software puede asegurar su funcionalidad durante muchos años, ahorrando tiempo y el coste económico de una migración total a una nueva aplicación.