UPC
Universidad Peruana de Ciencias Aplicadas

Unidad 6 – Prueba Unitarias

Introducción

UPC
Universidad Peruana de Ciencias Aplicadas

- Las pruebas son vitales en cualquier proyecto SW.
- Deben ser repetibles e incrementables.
- Introducimos un framework de pruebas unitarias.
- Útil también para otro tipo de pruebas: sistemas, integración.

Introducción

UPC
Universidad Peruana de Ciencias Aplicadas

- Un **programa** es **aceptable** cuando:
  - Hace lo que se acordó que debía hacer en las especificaciones.
  - No hace lo que no debe hacer.
- "Un programador **jamás** debería **entregar un programa sin haberlo probado**. Igualmente, quien recibe un programa de otro jamás debería aceptarlo sin haberlo probado.
- Para aprobar una práctica ésta debe pasar las pruebas funcionales.
- Cualquier **funcionalidad** de un programa **sin** una **prueba**
- automatizada, simplemente **no existe**" (Extreme Programming Explained", de Kent Beck)

DEFINICIONES

UPC
Universidad Peruana de Ciencias Aplicadas

¿Estamos construyendo correctamente el producto?

¿Estamos construyendo el producto correcto?

Verificación: El proceso de evaluación de un sistema (o de uno de sus componentes para determinar si los productos de una fase dada satisfacen las condiciones impuestas al comienzo de dicha fase.

Validación: El proceso de evaluación de un sistema o de uno de sus componentes durante o al final del proceso de desarrollo para determinar si satisface los requisitos marcados por el usuario.

Relación entre defecto – falla - error

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

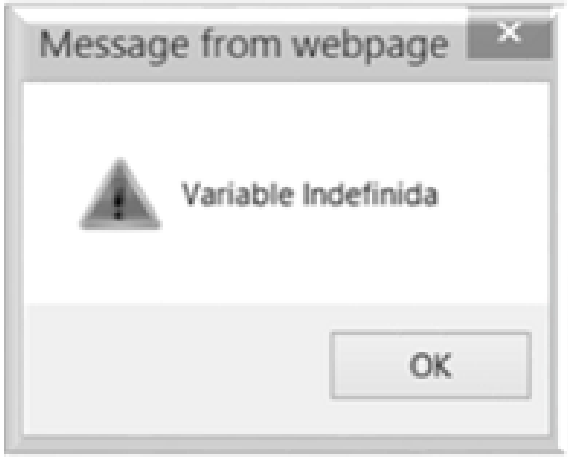
Error ("Error"):

- 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.**

Relación entre defecto – falla - error



Fallo ("Failure"):

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

Ideas paradójicas de las pruebas

UPC
Universidad Peruana de Ciencias Aplicadas

- **La prueba exhaustiva del software es impracticable** (no se pueden probar todas las posibilidades de su funcionamiento ni siquiera en programas sencillos).
- El objetivo de las pruebas es la detección de defectos en el software (descubrir un error es el éxito de una prueba).

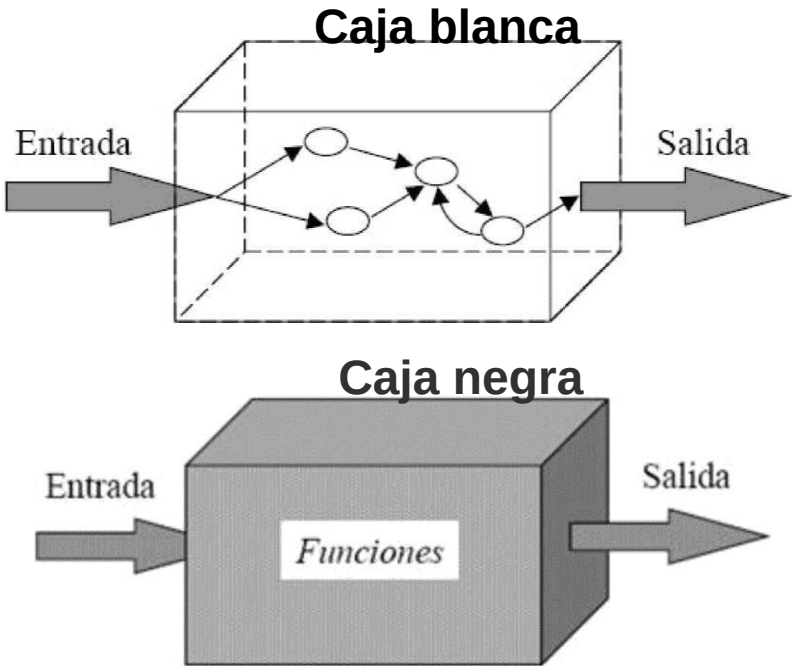
Mito: Un defecto implica que somos malos profesionales y que debemos sentirnos culpables.

Todo el mundo comete errores.

Enfoque de diseño de pruebas

Existen **tres** enfoques principales para el diseño de casos:

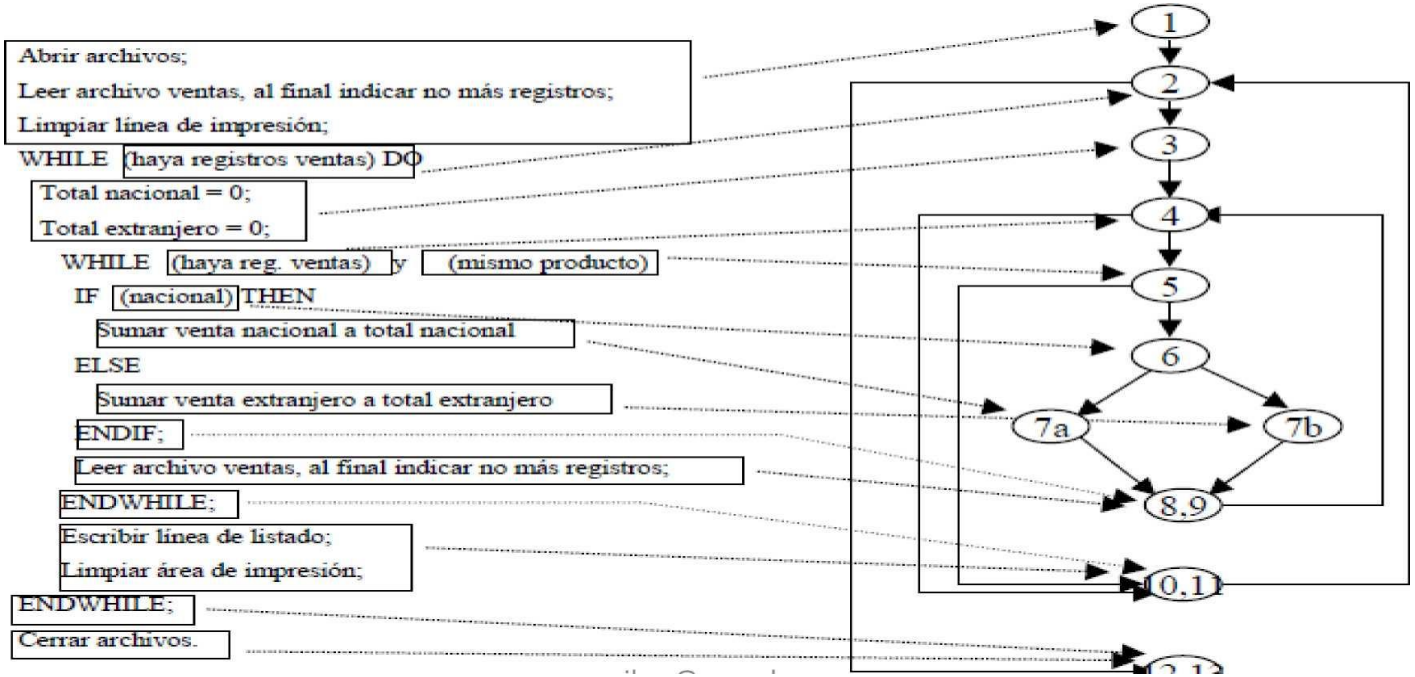
1. El enfoque **estructural** o de **caja blanca.** Se centra en la estructura interna del programa (analiza los caminos de ejecución).
2. El enfoque **funcional** o de **caja negra.** Se centra en las funciones, entradas y salidas.



3. El enfoque **aleatorio** consiste en utilizar modelos (en muchas ocasiones estadísticos) que representen las posibles entradas al programa para crear a partir de ellos los casos de prueba.

PRUEBAS ESTRUCTURALES

El diseño de casos de prueba tiene que estar basado en la elección de caminos importantes que ofrezcan una seguridad aceptable de que se descubren defectos (un programa de 50 líneas con 25 sentencias if en serie da lugar a 33,5 millones de secuencias posibles), para lo que se usan los criterios de cobertura lógica.



PRUEBAS FUNCIONALES

UPC
Universidad Peruana de Ciencias Aplicadas

- Se centran en las funciones, entradas y salidas. Seria impráctico probar el software para todas las posibilidades. De nuevo hay que tener criterios para elegir buenos casos de prueba.
- Un caso de prueba funcional es bien elegido si se cumple que:

  - Reduce el número de otros casos necesarios para que la prueba sea razonable. Esto implica que el caso ejecute el máximo número de posibilidades de entrada diferentes para así reducir el total de casos.
  - Cubre un conjunto extenso de otros casos posibles, es decir, no indica algo acerca de la ausencia o la presencia de defectos en el conjunto específico de entradas que prueba, así como de otros conjuntos similares.

PRUEBAS ALEATORIAS

UPC
Universidad Peruana de Ciencias Aplicadas

En las pruebas aleatorias simulamos la entrada habitual del programa creando datos de entrada en la secuencia y con la frecuencia con las que podrían aparecer en la práctica (de manera repetitiva).

Para ello habitualmente se utilizan generadores automáticos de casos de prueba.

ESTRATEGIAS DE APLICACIÓN DE LAS PRUEBAS

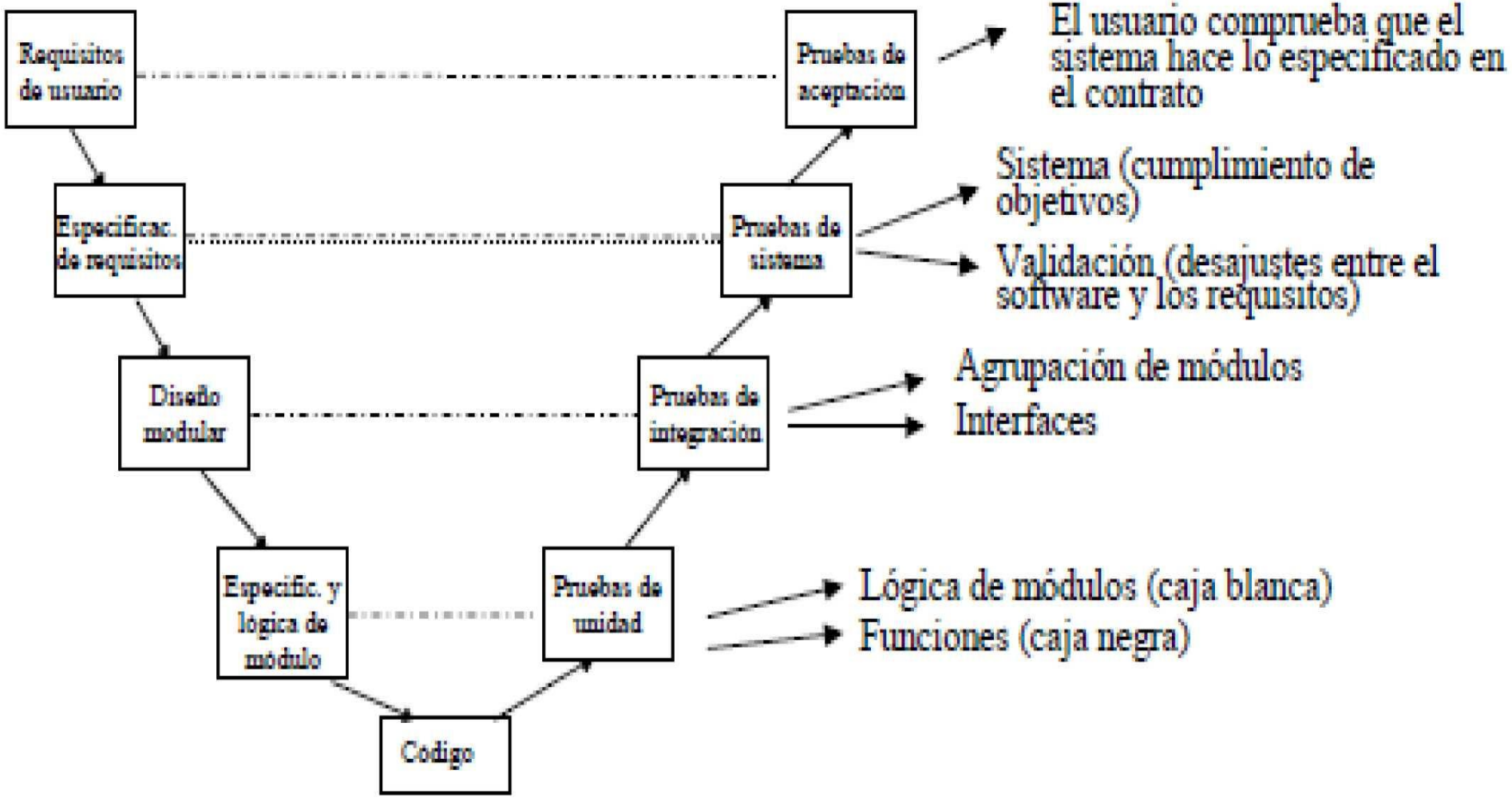
UPC
Universidad Peruana de Ciencias Aplicadas

- Se analiza como plantear las pruebas en el ciclo de vida del software.

- La estrategia de pruebas suele seguir estas etapas:

  - Comenzar pruebas a nivel de modulo
  - Continuar hacia la integración del sistema completo y su instalación
  - Culminar con la aceptación del producto por la parte del cliente

Relación entre productos del desarrollo y niveles de prueba



PRUEBAS DE UNIDAD

UPC
Universidad Peruana de Ciencias Aplicadas

Se trata de las pruebas formales que permiten declarar que un módulo está listo y terminado (no las informales que se realizan mientras se desarrollan los módulos) Hablamos de una **unidad de prueba** para referirnos a uno o más módulos que cumplen las siguientes condiciones [IEEE, 1986a]:

- Todos son del mismo programa
- Al menos uno de ellos no ha sido probado
- El conjunto de módulos es el objeto de un proceso de prueba

La prueba de unidad puede abarcar desde un módulo hasta un grupo de módulos (incluso un programa completo)

Estas pruebas suelen realizarlas el propio personal de desarrollo, pero evitando que sea el propio programador del módulo

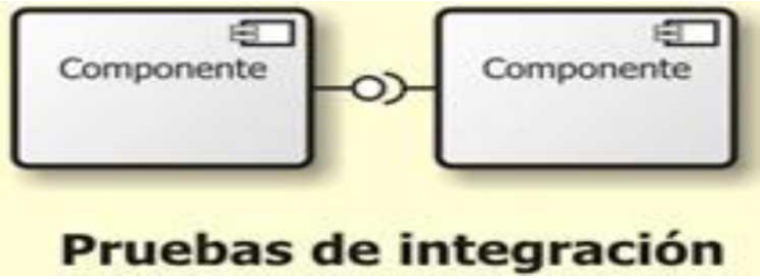
PRUEBAS DE INTEGRACIÓN

UPC
UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS

Implican una progresión ordenada de pruebas que van desde los componentes o módulos y que culminan en el sistema completo.

El orden de integración elegido afecta a diversos factores, como lo siguientes:

- La forma de preparar casos
- Las herramientas necesarias
- El orden de codificar y probar los módulos
- El coste de la depuración
- El coste de preparación de casos



PRUEBAS DE SISTEMA

Es el proceso de prueba de un sistema integrado de hardware y software para comprobar lo siguiente:

- Cumplimiento de todos los requisitos funcionales, considerando el producto software final al completo en un entorno de sistema.
- El funcionamiento y rendimiento en las interfaces hardware, software, de usuario y de operador.
- Adecuación de la documentación de usuario.
- Ejecución y rendimiento en condiciones límite y de sobrecarga.

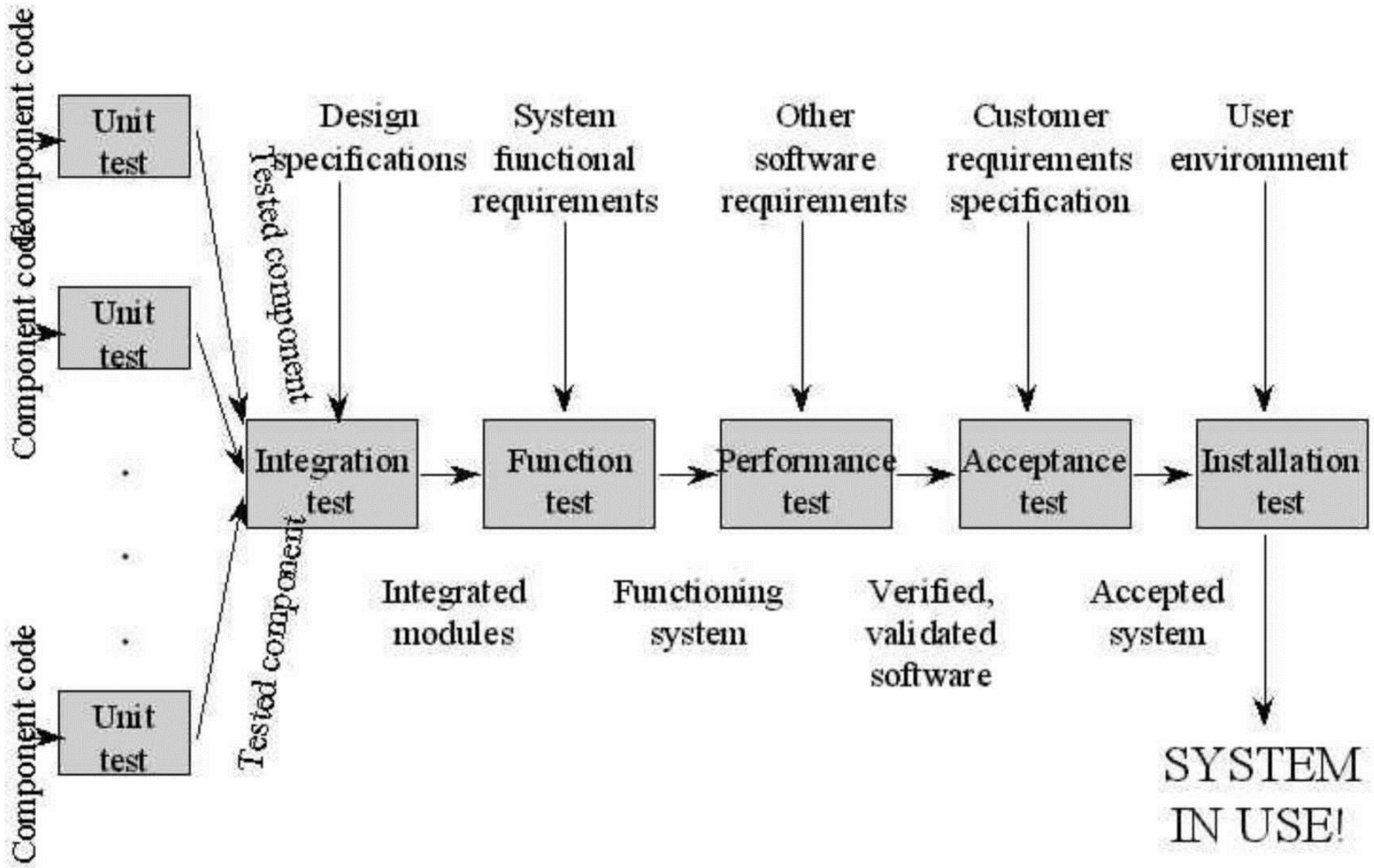
PRUEBAS DE ACEPTACION

Es la prueba planificada y organizada formalmente para determinar si se cumplen los requisitos de aceptación marcados por el cliente.

Sus características principales son las siguientes:

- Participación del usuario
- Está enfocada hacia la prueba de los requisitos de usuario especificados.
- Está considerada como la fase final del proceso para crear una confianza en que el producto es el apropiado para su uso en explotación

FLUJO


Component code
Component code
Component code
Unit test
Unit test
Unit test
Tested component
Tested component
Design specifications
System functional requirements
Other software requirements
Customer requirements specification
User environment
Integration test
Function test
Performance test
Acceptance test
Installation test
Integrated modules
Functioning system
Verified, validated software
Accepted system
SYSTEM IN USE!

Automatización de las pruebas

UPC
Universidad Peruana de Ciencias Aplicadas

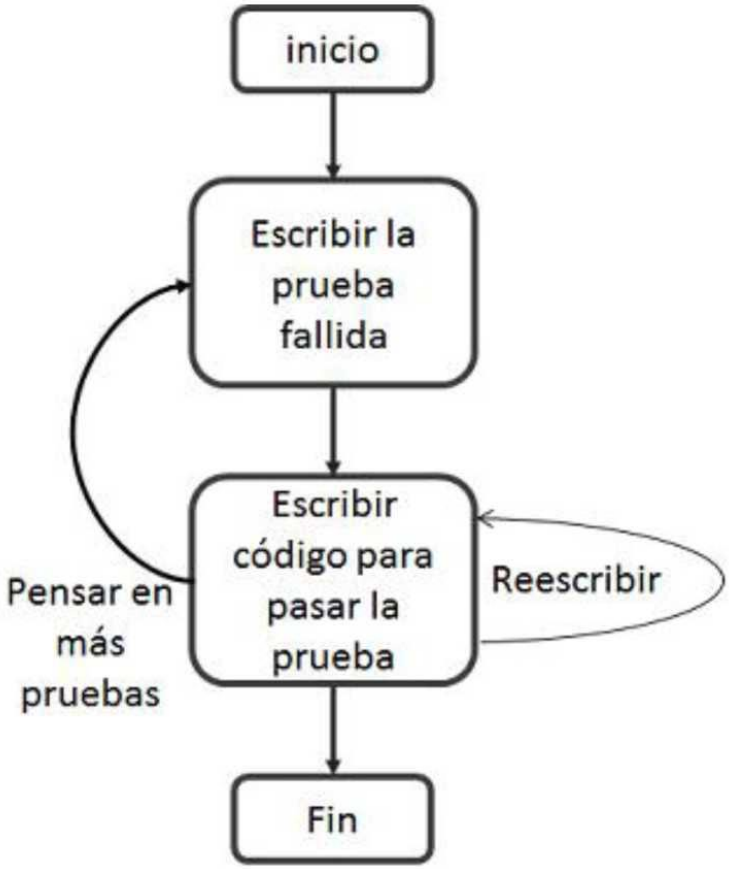
- Una vez que hemos diseñado todas las pruebas que cubrirán todos los criterios que serán evaluados, podemos proceder a su automatización en alguna herramienta de pruebas.
- Para tal objetivo existen algunas gratuitas con sus respectivas limitaciones, hasta aquellas más completas y/o complejas que se ofrecen para compra o en la nube, soportando diversas tecnologías, protocolos, sistemas operativos: Selenium, soapUI, TOSCA testsuite, HP Quick Test Pro, SilkTest, IBM Rational Functional Tester, SOATest, TestPartner, Visual Studio Test Profesional, etc.

Flujo de trabajo de Test-Driven Development (*TDD*)

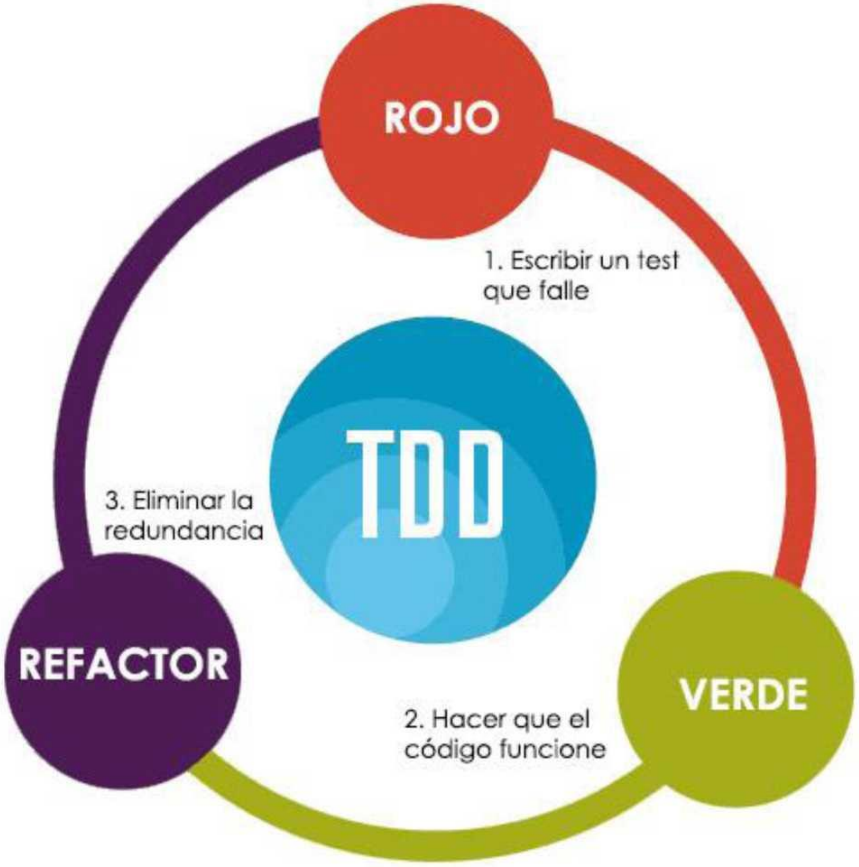
Lo importante en TDD es la palabra Dirigid lo cual indica que las pruebas dirigen el desarrollo de software.

El flujo de trabajo de TDD es:

- ✓ Escribir una prueba fallida
- ✓ Escribir código para hacer que pase la prueba
- ✓ Repetir los pasos 1 y 2
- ✓ Durante el proceso, re escribir el código agresivamente
- ✓ Cuando no pueda pensar en más pruebas, ha terminado



Flujo de trabajo de TDD



Ventajas del TDD

UPC
Universidad Peruana de Ciencias Aplicadas

- ✓ La calidad del software aumenta (y veremos por qué).
- ✓ Conseguimos código altamente reutilizable.
- ✓ El trabajo en equipo se hace más fácil, une a las personas. Nos permite confiar en nuestros compañeros aunque tengan menos experiencia.
- ✓ Multiplica la comunicación entre los miembros del equipo.
- ✓ Las personas encargadas de la garantía de calidad adquieren un rol más inteligente e interesante.
- ✓ Escribir el ejemplo (test) antes que el código nos obliga a escribir el mínimo de funcionalidad necesaria, evitando sobre diseñar.
- ✓ Los test's son la mejor documentación técnica que podemos consultar a la hora de entender qué misión cumple cada parte del software.
- ✓ Incrementa la productividad.
- ✓ Nos hace descubrir y afrontar más casos de uso en tiempo de diseño.
- ✓ La jornada se hace mucho más amena.
- ✓ Uno se marcha a casa con la reconfortante sensación de que el trabajo está bien hecho.

TDD vs. Pruebas al final

UPC
Universidad Peruana de Ciencias Aplicadas

- ✓ TDD insiste en que el desarrollo debe aparecer primero. Sólo cuando tenga su prueba escriba (y fallida) puede escribir el código para la prueba.
- ✓ Muchos programadores utilizan una variante escribiendo las pruebas en forma posterior, así primero escribe el código y luego las pruebas.
- ✓ En este caso, igual se realizan las pruebas, pero no obtiene los aspectos de diseño de TDD. Nada le impide escribir código no adecuado y luego invertir tiempo en escribir código para este código no adecuado. Escrito el código antes que las pruebas, uno tiene pre consideraciones de cómo el código debe funcionar, luego haces pruebas de esto.
- ✓ Con TDD se requiere de que el código haga lo contrario: escribir la prueba primero, y luego escribir el código para pasar la misma.

Recomendaciones para unas pruebas exitosas:

UPC
UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS

- ✓ Cada caso de prueba debe definir el resultado de salida esperado que se comparará con el realmente obtenido.
- ✓ El programador debe evitar probar sus propios programas, ya que desea (consciente o inconscientemente) demostrar que funcionan sin problemas. Además, es normal que las situaciones que olvidó considerar al crear el programa queden de nuevo olvidados al crear los casos de prueba.
- ✓ Se debe inspeccionar a conciencia el resultado de cada prueba, y así, poder descubrir posibles síntomas de defectos.
- ✓ Al generar casos de prueba, se deben incluir tanto datos de entrada válidos y esperados como no válidos e inesperados.

Recomendaciones para unas pruebas exitosas:

UPC
Universidad Peruana de Ciencias Aplicadas

- ✓ Las pruebas deben centrarse en dos objetivos (es habitual olvidar el segundo):
  - ❖ Probar si el software no hace lo que debe hacer
  - ❖ Probar si el software hace lo que debe hacer, es decir, si provoca efectos secundarios adversos.
- ✓ Se deben evitar los casos desechables, es decir, los no documentados ni diseñados con cuidado. Ya que suele ser necesario probar muchas veces el software y por tanto hay que tener claro qué funciona y qué no.
- ✓ No deben hacerse planes de prueba suponiendo que, prácticamente, no hay defectos en los programas y, por lo tanto, dedicando pocos recursos a las pruebas. Siempre hay defectos.

Recomendaciones para unas pruebas exitosas:

UPC
Universidad Peruana de Ciencias Aplicadas

- ✓ La experiencia parece indicar que donde hay un defecto hay otros, es decir, la probabilidad de descubrir nuevos defectos en una parte del software es proporcional al número de defectos ya descubierto.
- ✓ Las pruebas son una tarea tanto o más creativa que el desarrollo de software. Siempre se han considerado las pruebas como una tarea destructiva y rutinaria.

Es interesante planificar y diseñar las pruebas para poder detectar el máximo número y variedad de defectos con el mínimo consumo de tiempo y esfuerzo.