Universidad Nacional de Trujillo
Facultad de Ciencias Físicas y Matemáticas
Informática

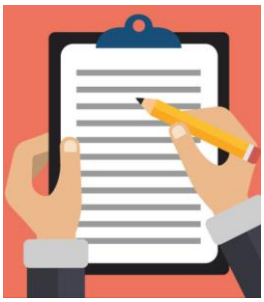
Sesión 12
Pruebas del Software

Logro de Aprendizaje.

Al finalizar la Sesión, el estudiante realiza las pruebas del software de su proyecto integrador.

Contenido

1. Pruebas del software
2. Pruebas de caja negra
3. Pruebas de caja blanca



Pruebas de software.

- Las pruebas de software son una parte importante pero muy costosa del proceso de desarrollo de software
- Pueden llegar a representar entre el 30 y 50% del costo total del desarrollo del software [Myers, 2004]
- Sin embargo, los costos de las fallas en un software en operación pueden llegar a ser mucho mayores (catastróficos)

"La mayoría del software actual es muy parecido a una pirámide egipcia, con millones de ladrillos puestos unos encima de otros sin una estructura integral, simplemente realizada a base de fuerza bruta y miles de esclavos"
-- *Alan Kay*

Fallos de software.

¿Has tendido una experiencia de fallo del software?

Fallos de software.

La catástrofe del Hartford Coliseum (1978). 70 millones de dólares.

Apenas unas horas después de que miles de aficionados abandonaron el Hartford Coliseum, el techo se derrumbó por el peso de la nieve. La causa: cálculo incorrecto introducido en el software CAD utilizado para diseñar el coliseo.

Fallos de software.

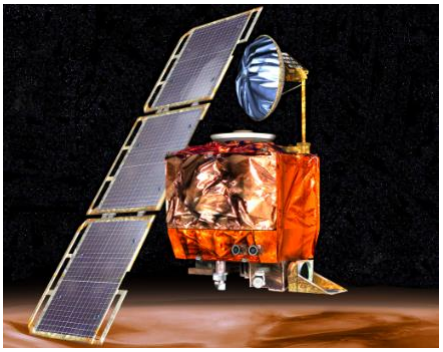
Explosión del cohete Arian (1996). 500 millones de dólares.

En el 1996, el cohete Ariane 5 de la Agencia Espacial Europea estalló. El Ariane explotó porque un número real de 64 bits (coma flotante) relacionado con la velocidad se convirtió en un entero de 16 bits.

Fallos de software.

Mars Climate Orbiterm (1999). 655 millones de dólares.

En 1999 los ingenieros de la NASA perdieron el contacto con la Mars Climate Orbiter en su intento que orbitase en Marte. La causa, un programa calculaba la distancia en unidades inglesas (pulgadas, pies y libras), mientras que otro utilizó unidades métricas.



Fallos de software.

En el año 2000 hubo una sobredosis radiológica en el Instituto Nacional del Cáncer de Panamá, los ingenieros de la empresa Multidata Systems International calcularon erróneamente la dosis de radiación que un paciente podría recibir durante la terapia de radiología. El fallo estaba en el software de control de la máquina de rayos, lo que provocó que al menos ocho pacientes murieran por las altas dosis recibidas y otros 20 tuvieran problemas de salud graves.

Error, defecto, fallo.

Error:

Una equivocación de una persona al desarrollar alguna actividad de desarrollo de software

Defecto:

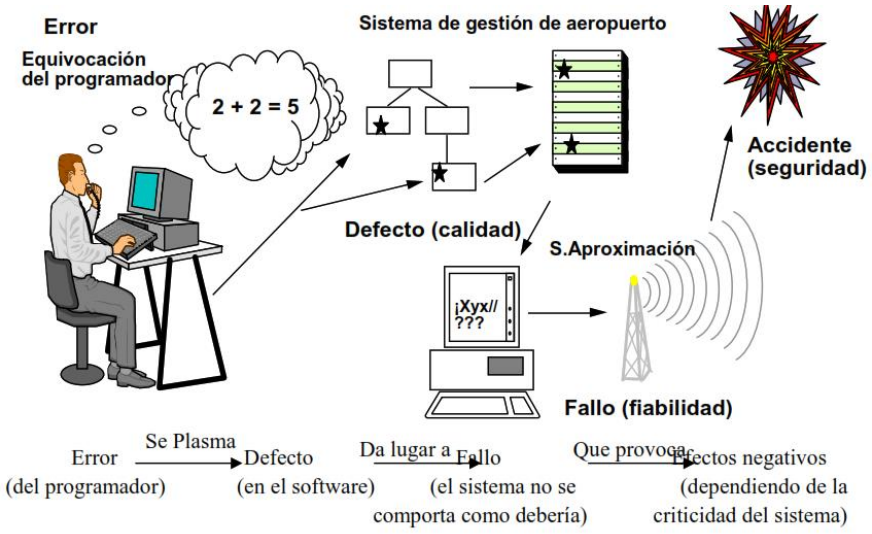
Se produce cuando una persona comete un error

Falla:

Es un desvío respecto del comportamiento esperado del sistema.

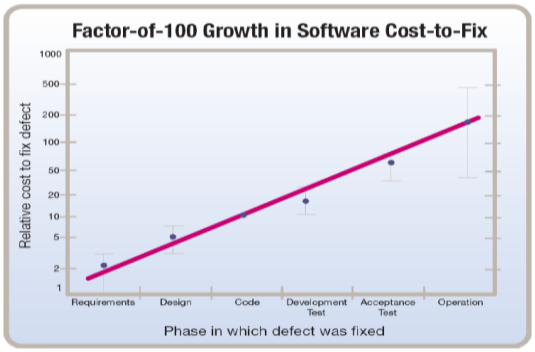
Defecto es una vista interna, lo ven los desarrolladores. Falla es una vista externa, la ven los usuarios.

Error, defecto, fallo.



El costo de los defectos.

- Los costos de eliminar defectos se incrementan con el tiempo durante el cual el defecto permanece en el sistema.
- La detección de errores en etapas tempranas permite la corrección de los mismos a costos reducidos.



¿Por qué probar el software?

- **Mejora de la calidad de un producto software.** El proceso de pruebas ayuda a suministrar o aportar al software los atributos deseados, por ejemplo: retirar defectos que conducen a fallos.
- **Reducción del riesgo de detectar errores.** Las actividades de pruebas de software adecuadas reducirán el riesgo de encontrar errores durante la fase de operación del software.
- **Satisfacer compromisos.** La ejecución de pruebas puede ser un requisito obligatorio por parte del cliente, debido a normas legales, así como al cumplimiento de estándares propios de una industria.

Las pruebas son una inversión

Prueba de software.

Algunas definiciones incorrectas:

- Probar es demostrar que no hay errores presentes en un programa.
- El propósito de probar es mostrar que el programa realiza correctamente las funciones esperadas.

La definición Correcta

- Probar es el proceso de ejecución de un programa con el fin de encontrar errores.

Prueba exitosa:

- Aquella que detecta errores

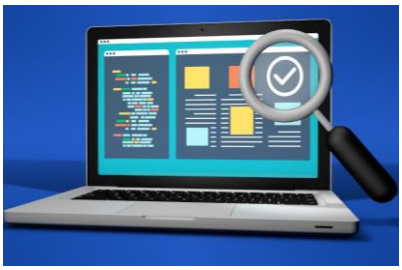
Prueba no exitosa:

- Aquella que no detecta errores

Prueba de software.

- Una prueba es una actividad en la que un sistema o un componente es ejecutado bajo condiciones especificadas, los resultados son observados o registrados, y una evaluación es realizada de un aspecto del sistema o componente. [IEEE Std.610.12-1990]

- Una prueba es una actividad ejecutada para evaluar y mejorar la calidad del producto a través de la identificación de defectos y problemas. [SWEBOK]



Prueba de software.

- La prueba es el proceso de ejecución de un programa con la intención de descubrir un error.
- Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
- Una prueba tiene éxito si descubre un error no detectado hasta entonces.

Principios de las pruebas.

- A todas las pruebas se les debería poder hacer un seguimiento hasta los requisitos del cliente.
- Las pruebas deberían planificarse mucho antes de que empiecen.
- Las pruebas deberían empezar por “lo pequeño” y progresar hacia “lo grande”.
- No son posibles las pruebas exhaustivas.
- Para ser más eficaces (pruebas con la más alta probabilidad de encontrar errores), las pruebas deberían ser realizadas por un equipo independiente.
- Cada caso de prueba debe definir el resultado de salida esperado.
- Al generar casos de prueba, se deben incluir tanto datos de entrada válidos y esperados como no válidos e inesperados.

Niveles de pruebas.

- Llamado también estrategias o etapas de prueba de software.
- Especifican en qué momento del desarrollo del software se realizan las pruebas.
- Cada nivel tiene un objetivo diferente, según el momento son apropiadas diferentes técnicas de prueba.
- Se comienza probando las partes más pequeñas y se continua con las más grandes.
- Son:
  - Pruebas de unidad
  - Pruebas de integración
  - Pruebas de validación
  - Pruebas del sistema

Técnicas (Métodos) de pruebas

- **Estáticas**

  Es una técnica de prueba de software que se utiliza para comprobar si hay defectos en el rendimiento del software sin ejecutar el código.

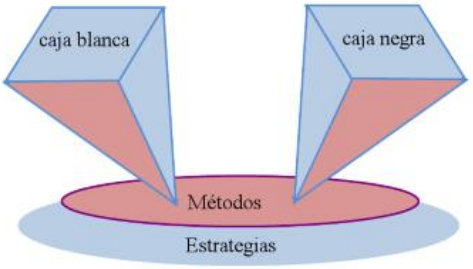
  Se realizan para evitar errores en una etapa temprana de su desarrollo porque es más fácil identificar los errores y resolverlos.

- **Dinámicas**

  Verifica una aplicación cuando se ejecuta el código

Técnicas (Métodos) de pruebas

- **Pruebas estáticas:**
  - Revisiones formales e informales
  - Análisis estático
- **Pruebas dinámicas**
  - Pruebas de caja blanca (Estructurales)
  - Pruebas de caja negra (Funcionales)



Criterios para completar la prueba.

Cada vez que se tratan de las pruebas del SW surgen unas preguntas clásicas:

- ¿Cuándo hemos terminado la prueba?
- ¿Cómo sabemos que hemos probado lo suficiente?

- '¿Cuando debemos probar?'
- Una respuesta a la "*La prueba nunca termina ya que el responsable carga o pasa el problema al cliente*"
- Otra respuesta algo cínica es "*Se termina la prueba cuando se agota el tiempo o el dinero disponible para cada efecto*"

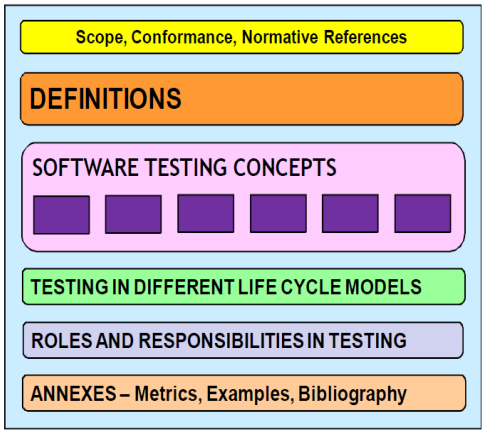
ISO/IEC/IEEE 29119.

- En mayo de 2007 la ISO formó un grupo de trabajo para desarrollar un estándar de prueba de software.
- En setiembre de 2013 se publica los tres primeros estándares.
- En octubre de 2021se publica la segunda edición.
- El Estándar de prueba ISO/IEC/IEEE 29119 está formado por 5 partes en draft:
  - Parte 1: Conceptos y Terminología
  - Parte 2: Procesos de Prueba
  - Parte 3: Documentación de Prueba
  - Parte 4: Técnicas de Prueba
  - Parte 5: Pruebas conducidas por palabras clave

ISO/IEC/IEEE 29119.

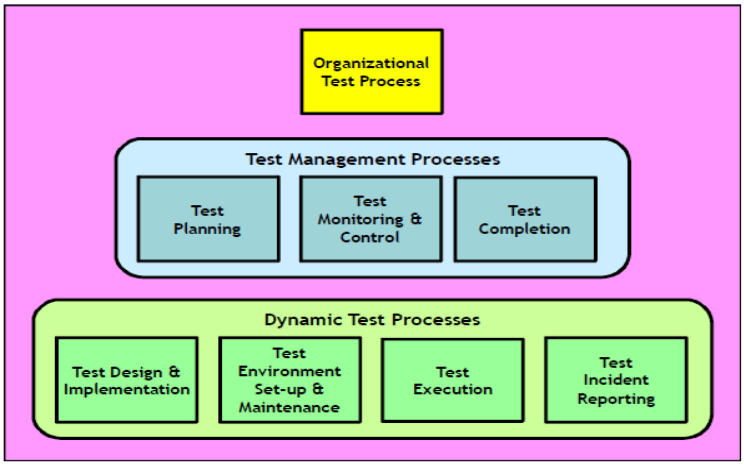
Parte 1: Conceptos y Terminología

Da una introducción a los estándares y definiciones de pruebas.



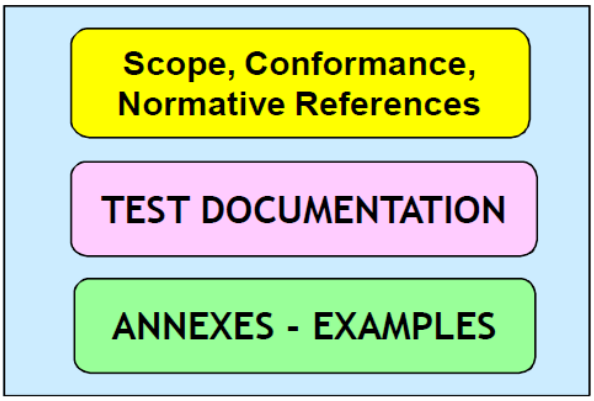
ISO/IEC/IEEE 29119.

Parte 2: Procesos de Prueba



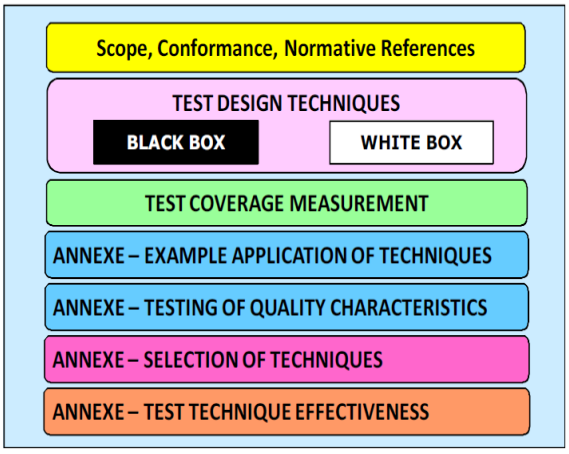
ISO/IEC/IEEE 29119.

Parte 3: Documentación de Prueba



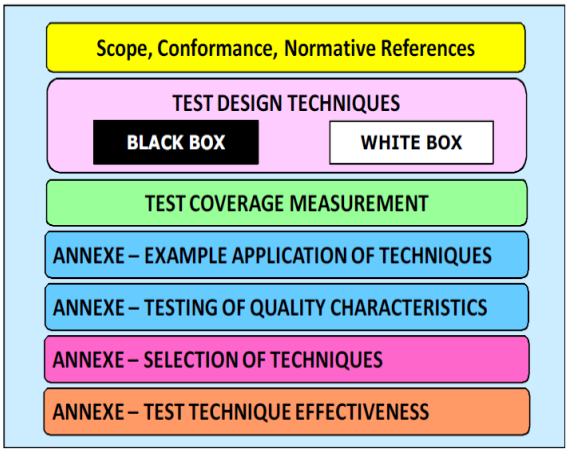
ISO/IEC/IEEE 29119.

Parte 4: Técnicas de Prueba



ISO/IEC/IEEE 29119.

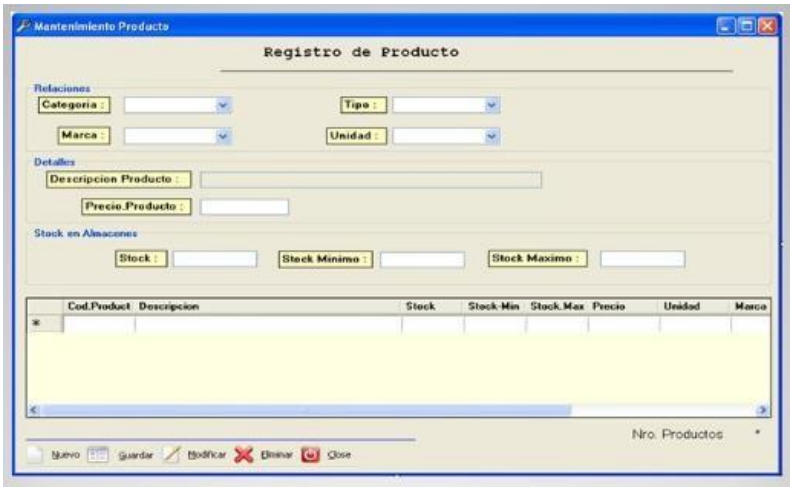
Parte 4: Técnicas de Prueba



Prueba de caja negra.

- Es un método de prueba de software que analiza la funcionalidad de un software / aplicación sin saber mucho sobre la estructura / diseño interno del elemento que se está probando y compara el valor de entrada con el valor de salida.
- También se llama prueba de comportamiento, de caja cerrada, basada en especificaciones.
- La prueba de Caja Negra se centra principalmente en los requisitos funcionales del software.

Prueba de caja negra.



Partición de equivalencia.

- También se conoce como Partición de clases de equivalencia (ECP).
- En esta técnica, los valores de entrada a la aplicación se dividen en diferentes clases o grupos según su similitud en el resultado.
- Por lo tanto, en lugar de usar todos y cada uno de los valores de entrada, ahora podemos usar cualquier valor del grupo / clase para probar el resultado. De esta manera, podemos mantener la cobertura de la prueba mientras podemos reducir una gran cantidad de retrabajos y, lo más importante, el tiempo invertido.
- Ejemplo:

NOTA [ ] (valores aceptados de 0 a 20)

INVALIDA	VALIDA	INVALIDA
< 0	0 - 20	>20

Partición de equivalencia.

Pasos para identificar clases de equivalencia.

1. Identificación de las condiciones de entrada del programa.
2. Identificar las clases de equivalencia:
   a) Datos válidos.
   b) Datos no válidos.
3. Asignar un número único para cada clase de equivalencia.
4. Escribir casos de pruebas para todas las clases válidas.
5. Escribir casos de pruebas para todas las clases no válidas.

Partición de equivalencia - Ejemplo.

Ejemplo: El ingreso de una nota

NOTA [ ] (valores aceptados de 0 a 20)

ENTRADA	CLASES DE EQUIVALENCIA			
	VALIDA	INVALIDA	INVALIDA	INVALIDA
Enteros de 0 a 20	1) 0 - 20	2) < 0	3) >20	4) No es número

Casos de prueba:

CLASE DE EQUIVALENCIA	DATOS DE PRUEBA	RESULTADO ESPERADO	RESULTADO REAL	SITUACIÓN (OK/ERROR)
1)	4	ACEPTADO	ACEPTADO	OK
2)	-5	RECHAZADO	ACEPTADO	ERROR
3)	23	RECHAZADO	RECHAZADO	OK
4)	HOLA	RECHAZADO	RECHAZADO	OK

Análisis de valores límite.

- En esta técnica hay que enfocarse en los valores en los límites, ya que se encuentra que muchas aplicaciones tienen una gran cantidad de problemas en los límites.
- Si la condición de entrada especifica un número finito y consecutivo de valores, escribir casos de prueba para los valores máximo, mínimo, uno más del máximo y uno menos del mínimo.

Análisis de valores límite.- Ejemplo.

Ejemplo: El ingreso de una nota

NOTA [ ] (valores aceptados de 0 a 20)

ENTRADA	VALORES LÍMITE			
	VALIDA	VALIDA	INVALIDA	INVALIDA
Enteros de 0 a 20	1) 0	2) 20	3) -1	4) 21

Casos de prueba:

CLASE DE VALOR LÍMITE	DATOS DE PRUEBA	RESULTADO ESPERADO	RESULTADO REAL	SITUACIÓN (OK/ERROR)
1)	0	ACEPTADO	ACEPTADO	OK
2)	20	ACEPTADO	ACEPTADO	OK
3)	-1	RECHAZADO	RECHAZADO	OK
4)	21	RECHAZADO	ACEPTADO	ERROR

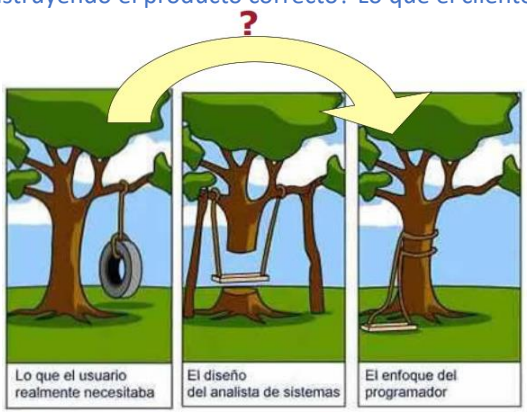
Ejercicio.

Codificar el ingreso de una nota en un lenguaje de programación y realizar las pruebas de caja negra, prueba partición de equivalencia y análisis de valores límite

NOTA [ ] (valores aceptados de 0 a 20)

Prueba de validación.

- La validación puede definirse de muchas formas, pero una simple definición es que la validación se consigue cuando el software funciona de acuerdo con las expectativas razonables del cliente.
- ¿Estamos construyendo el producto correcto? Lo que el cliente quiere.



Prueba de validación.

- ¿Estamos construyendo el producto correcto? Lo que el cliente quiere.

Prueba de validación.

- Se tienen las siguientes pruebas de validación:
  - La *prueba alfa* se lleva a cabo, por un cliente, en el lugar de desarrollo. Se usa el software de forma natural con el desarrollador como observador del usuario y registrando los errores y los problemas de uso. Las pruebas alfa se llevan a cabo en un entorno controlado.
  - La *prueba beta* se lleva a cabo por los usuarios finales del software en los lugares de trabajo de los clientes. A diferencia de la prueba alfa, el desarrollador no está presente normalmente. Así, la prueba beta es una aplicación «en vivo» del software en un entorno que no puede ser controlado por el desarrollador.

Prueba de validación.

Banco CCD S.A.
Departamento de Informática

Acta de Validación

Sistema: Créditos
Módulo: Análisis de Clientes aptos para créditos
Lugar de la Prueba: Departamento de Informática-Unidad de Desarrollo
Fecha: 25/11/2014 **Hora:** 4:00pm – 4:30pm

	Observaciones:		**Conformidad**

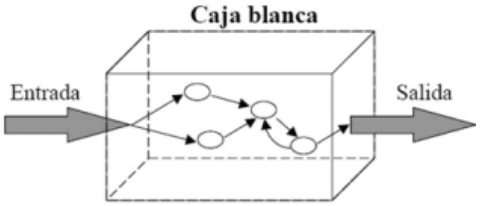
Responsables:

Ing. Luis Pérez Tejada
Desarrollador de Software

CPC María Flores Moreno
Analista de Créditos

Prueba de caja blanca.

- La prueba de la caja blanca es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para derivar los casos de prueba.
- También se llama prueba de caja de cristal.



Prueba de caja blanca.

Intentan garantizar que:

- Se ejecutan al menos una vez todos los caminos independientes de cada módulo.
- Se utilizan las decisiones en su parte verdadera y en su parte falsa.
- Se ejecuten todos los bucles en sus límites.
- Se utilizan todas las estructuras de datos internas.

Prueba de caja blanca.

Intentan garantizar que:

- Se ejecutan al menos una vez todos los caminos independientes de cada módulo.
- Se utilizan las decisiones en su parte verdadera y en su parte falsa.
- Se ejecuten todos los bucles en sus límites.
- Se utilizan todas las estructuras de datos internas.

Prueba de caja blanca.

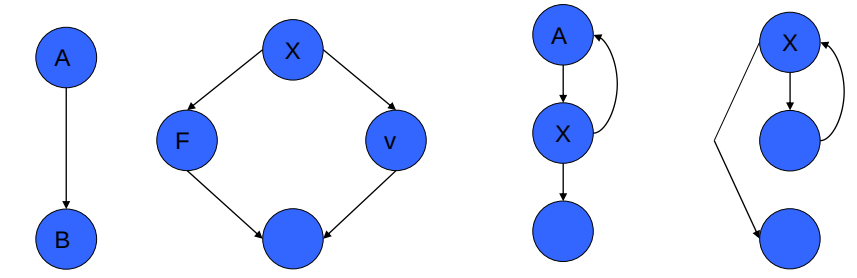
Se tienen las siguientes técnicas:

- Prueba del camino básico.
- Prueba de condición.
- Prueba de flujo de datos.
- Prueba de bucles.

Prueba del camino básico.

El método del camino básico (propuesto por McCabe) permite obtener una medida de la complejidad de un diseño procedimental, y utilizar esta medida como guía para la definición de una serie de caminos básicos de ejecución, diseñando casos de prueba que garanticen que cada camino se ejecuta al menos una vez.

Notación del grafo de flujo o grafo del programa



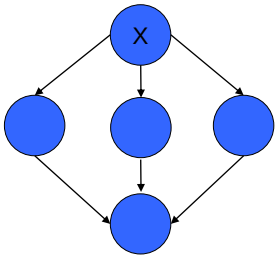
Secuencia

Si X Entonces V Sino F…
(If X Then V Else F)

Hacer A Mientras X
(Do A While X)

Mientras X
(While X)

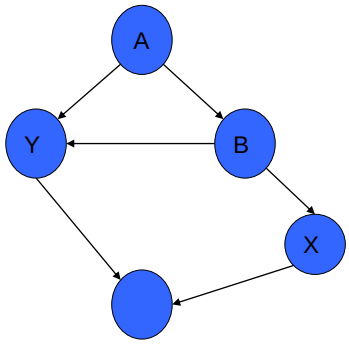
Notación del grafo de flujo o grafo del programa



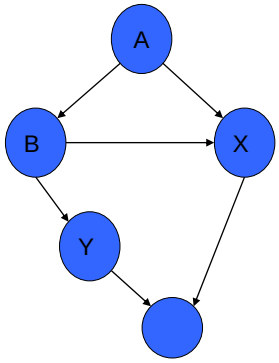
Según sea X
(Case X)

- En el grafo de flujo
  1. Cada nodo representa una o más sentencias procedimentales
  2. Un solo nodo puede corresponder a una secuencia de pasos del proceso y a una decisión
  3. Las flechas (aristas) representan el flujo de control
- Cualquier representación del diseño procedimental se puede traducir a un grafo de flujo.

Notación del grafo de flujo o grafo del programa

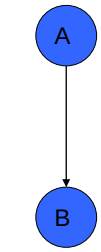


IF A && B ENTONCES
    X
SINO
    Y



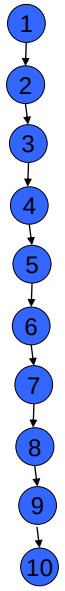
IF A || B ENTONCES
    X
SINO
    Y

Notación del grafo de flujo o grafo del programa

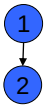


Secuencia

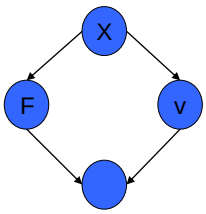
```
#include <iostream> 1

using namespace std;  2

int main(void)   3
{
int N1,N2;    4
cout<<"Ingrese primer número:";  5
cin>>N1;   6
cout<<"Ingrese segundo número:";   7
cin>>N2;   8
cout<<"La suma es "<<N1+N2;   9
return(0);    10
}
```



Equivale a

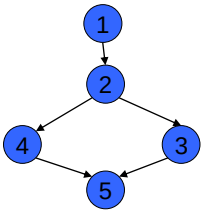


Notación del grafo de flujo o grafo del programa

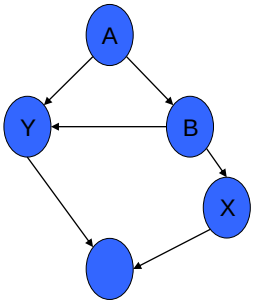


Si X Entonces V Sino F…
(If X Then V Else F)

```
#include <iostream>  1

using namespace std;  1

int main(void)  1
{
int N;  1
cout<<"Ingrese un número:";  1
cin>>N;  1
if (N % 2 == 0)   2
 cout<<"Es par";   3
else
 cout<<"Es impar";   4
return(0);  5
}
```

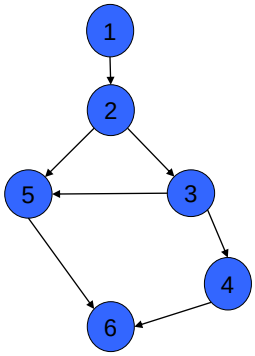


Notación del grafo de flujo o grafo del programa

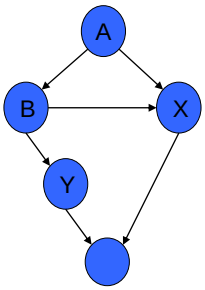


IF A && B ENTONCES
X
SINO
Y

```
#include<iostream>  1

using namespace std;  1

int main(void)  1
{
int N;  1

cout<<"Ingrese una nota:";  1
cin>>N;  1
      2            3
if (N>=0 && N<=20)
   cout<<"Es una nota correcta";  4
else
   cout<<"Es una nota incorrecta";  5
return(0);  6
}
```

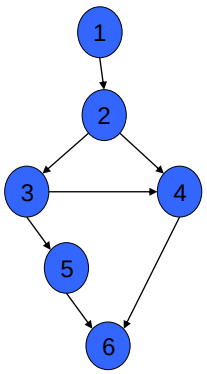


Notación del grafo de flujo o grafo del programa

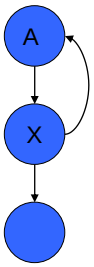


IF A || B ENTONCES
X
SINO
Y

```
#include<iostream>  1

using namespace std;  1

int main(void)  1
{
int N;  1

cout<<"Ingrese una nota:";  1
cin>>N;  1
      2            3
if (N<0 || N>20)
   cout<<"Es una nota incorrecta";  4
else
   cout<<"Es una nota correcta";  5
return(0);  6
}
```

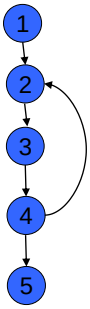


Notación del grafo de flujo o grafo del programa

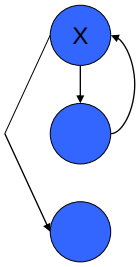


Hacer A Mientras X
(Do A While X)

```
#include <iostream>  1

using namespace std;  1

int main(void)  1
{
int N;  1
do {  2
  cout<<"Ingrese un número positivo:";  3
  cin>>N;  3
} while (N<1);  4
cout<<"El numero es positivo";  5
return(0);  5
}
```

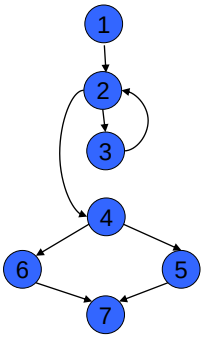


Notación del grafo de flujo o grafo del programa

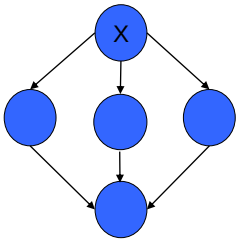


Mientras X
(While X)

```
#include <iostream> 1
#include <string.h>  1

using namespace std;  1

int main(void)  1
{
int I=0;  1
char N[20];  1
cout<<"Ingrese un número entero:";  1
cin>>N;  1
while (isdigit(N[I]))  2
   I++;  3
if (strlen(N)==I)  4
  cout<<"El numero es entero";  5
else
  cout<<"ERROR: No es un numero entero";  6
return(0);  7
}
```

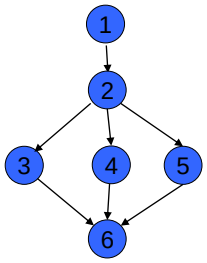


Notación del grafo de flujo o grafo del programa



Según sea X
(Case X)

```
#include <iostream>  1

using namespace std; 1

int main(void)  1
{
char S;  1
cout<<"Sexo M=Masculino, F=Femenino:";  1
cin>>S;  1
switch (S)  2
 {  case 'M':  3
      cout<<"Eres varon";  3
      break;  3
  case 'F':  4
      cout<<"Eres mujer";  4
      break;  4
   default :  5
      cout<<"Ingresaste una opcion incorrecta";  5
 }
return(0);  6
}
```



Ejercicio

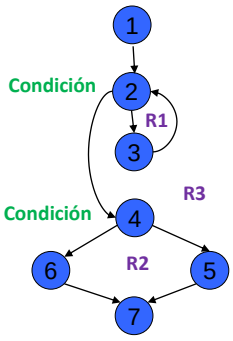
Dibujar el grafo de flujo del siguiente programa.

```
#include <iostream>
using namespace std;
int main(void)
{
int N;
int SUMA=1,I=2;
cout<<"Ingrese un numero entero:";
cin>>N;
while (I<=N/2)
  { if (N%I==0)
      SUMA=SUMA+I;
    I++;
  }
if (N==SUMA)
   cout<<"Es numero perfecto";
else
   cout<<"No es numero perfecto";
  return(0);
}
```

Complejidad ciclomática

- Es una medida del software que proporciona una medición cuantitativa de la complejidad lógica de un programa.
- Cuando se usa en el contexto del método del prueba del camino básico, el valor calculado como complejidad ciclomática define el número de ***caminos independientes*** del conjunto básico de un programa y nos indica el número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez.

Cálculo de la complejidad ciclomática



1. V (G) = A – N + 2
2. V (G) = R
3. V (G) = C + 1

Donde

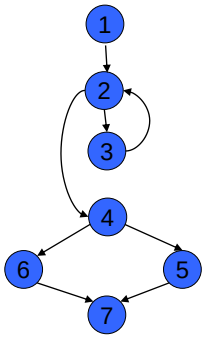
A : # de arcos o aristas del grafo.
N : # de nodos.
R : # de regiones cerradas del grafo.
C : # de nodos de condición.

V(G) = 8 – 7 + 2 = **3**

V(G) = 3 regiones

V(G) = 2 + 1 = 3

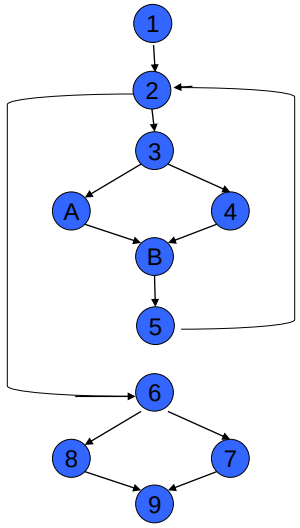
Camino independiente



- Un ***camino independiente*** es cualquier camino del programa que introduce, por lo menos, un nuevo conjunto de sentencias de proceso o una nueva condición.
- En términos del gafo de flujo, un camino independiente está constituido por lo menos por una arista que no haya sido recorrida anteriormente a la definición del camino.
- Para el grafo de flujo, un conjunto de caminos independientes sería:
  a) 1,2,4,5,7
  b) 1,2,4,6,7
  c) 1,2,3,4,5,7
- El siguiente camino 1,2,3,4,6,7 no se considera un camino independiente, ya que es simplemente una combinación de caminos ya especificados y no recorre ninguna nueva arista.

Ejercicio

Determinar la complejidad ciclomática del grafo de flujo e indicar un conjunto de caminos independientes.



Prueba del camino básico.

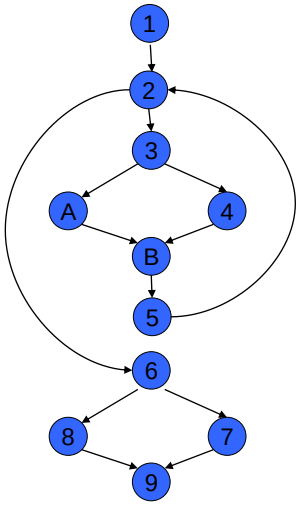
Para el diseño de pruebas mediante el camino básico, se siguen los siguientes pasos:

1. Dibujar el grafo de flujo, a partir del código del módulo
2. Determinar la complejidad ciclomática del grafo de flujo
3. Definir un conjunto de caminos independientes
4. Diseñar los casos de prueba que permitan la ejecución de cada uno de los caminos anteriores
5. Ejecutar cada caso de prueba y comprobar que los resultados son los esperados.

Prueba del camino básico - Ejemplo

```
#include <iostream>  1
using namespace std;  1
int main(void)  1
{
int N;  1
int SUMA=1,I=2;  1
cout<<"Ingrese un numero entero:";  1
cin>>N;  1
while (I<=N/2)  2
 { if (N%I==0)  3
    SUMA=SUMA+I;  4
   I++;  5
 }
if (N==SUMA)  6
  cout<<"Es numero perfecto";  7
else
  cout<<"No es numero perfecto";  8

return(0);  9
}
```

1. Dibujar su grafo de flujo

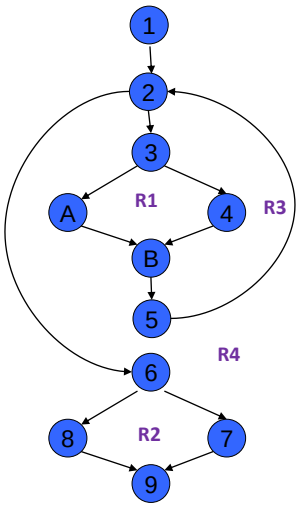


Prueba del camino básico

2. Determinar la complejidad ciclomática

   V(G) = A – N + 2
   = 13 – 11 + 2
   = 4

3. Definir un conjunto de caminos independientes
   a) 1,2,6,7,9
   b) 1,2,6,8,9
   c) 1,2,3,4,B,5,2,…
   d) 1,2,3,A,B,5,2,…



Prueba del camino básico

4. Diseñar los casos de prueba

CAMINO	VALORES DE ENTRADA	RESULTADO ESPERADO	RESULTADO REAL	SITUACIÓN (OK/ERROR)
a)1,2,6,7,9	N=1	No es número perfecto	Es número perfecto	ERROR
b)1,2,6,8,9	N=2	No es número perfecto	No es número perfecto	OK
c)1,2,3,4,B,5,…	N=8	No es número perfecto	No es número perfecto	OK
	N=6	Es número perfecto	Es número perfecto	Ok
d)1,2,3,A,B,5,2,.	N=7	No es número perfecto	No es número perfecto	OK

5. Ejecutar cada caso de prueba y comprobar que los resultados son los esperados

Prueba del camino básico

Cuando V (G) >10 la probabilidad de errores en el módulo o programa crece mucho entonces se recomienda dividir el módulo.

Conclusiones.

- Las pruebas de software es un proceso complejo, dentro del proceso de desarrollo, que debe llevarse a cabo para asegurar la calidad del software.
- El proceso de pruebas es de vital importancia dentro del ciclo de vida del software, de esta manera se puede verificar la calidad del producto antes de su puesta en producción.

Tarea

En una aplicación bancaria un cliente debe ingresar su código de cliente, una clave y nombre de la operación. El código de cliente es un numero de 6 dígitos, la clave es un número de 4 dígitos y la operación puede ser: “TRANSFERENCIA”, “PAGO DE SERVICIO”, “DEPÓSITO” y RETIRO”.

Responder las siguientes preguntas:

1. Identificar las entradas, las clases de equivalencia y diseñar sus casos de prueba
2. Identificar las entradas, las clase de valores límites y diseñar sus casos de prueba

Tarea

Ingresar a venta de pasajes online de ITTSA BUS:
https://www.ittsabus.com/online/

Y simular la compra de un pasaje.

En la ventana ***REGISTRA PASAJEROS,*** identificar las entradas, las clases de equivalencia y diseñar sus casos de prueba para probar los campos ***N° documento y Nombres***