Planificación TERCER Trimestre

Fecha/tipo	Actividad
13 Abril 2021	Programación trimestral **Udt 5. Diseño y realización de pruebas** ▪ Pruebas de caja blanca. ▪ Pruebas de caja negra
20 Abril 2021	
27 Abril 2021	▪ Pruebas Unitarias con JUnit ▪ Control de versiones Git y GitHub
4 Mayo 2021	
11 Mayo 2021	**Udt 6. Optimización y Documentación** ▪ Refactorización ▪ Documentación con JavaDoc
18 Mayo 2021	
25 Mayo 2021??	▪
Consultar Web o Tutoría Campus	**EXAMEN TRIMESTRAL**
Consultar Web o Tutoría Campus	**EXAMEN FINAL**

Diseño y realización de pruebas

Objetivos

Conocer la importancia de la realización de pruebas, durante el proceso de desarrollo de software.

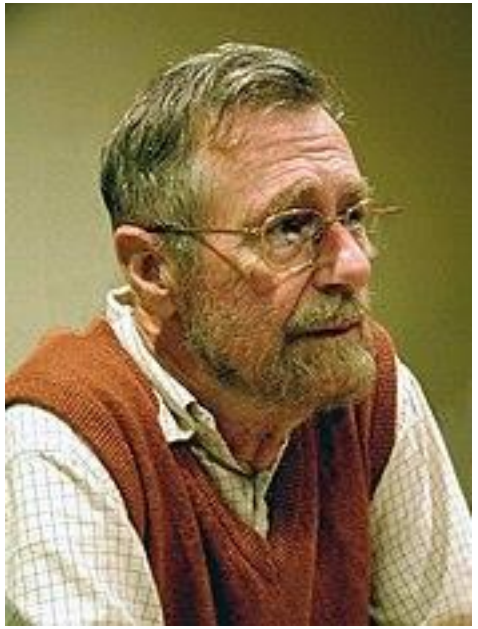
Definir casos de prueba.

Utilizar herramientas de depuración en un entorno de desarrollo.

Realizar pruebas unitarias utilizando la herramienta JUnit.

Implementar pruebas automáticas.

La prueba
es el *proceso* de
ejecutar un programa
con el fin de
ENCONTRAR
ERRORES



Wikipedia. Edsger Dijkstra en 2002

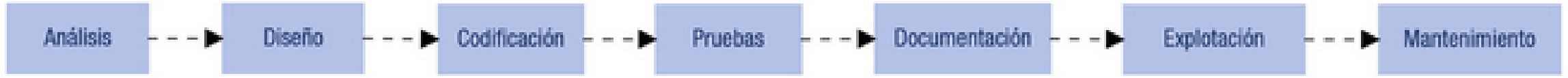
Las pruebas solo pueden demostrar la presencia de errores, no su ausencia.

Ciclo de Vida

Proceso que incluye todos los pasos que se dan desde que se concibe una idea hasta que una aplicación informática está implementada en el ordenador y funcionando.

Pasos:

- **Análisis**: especificación de requisitos funcionales (comportamiento) y no funcionales (tiempos, legislación).
- **Diseño**: división del sistema en partes y sus relaciones.
- **Codificación**: elegir lenguaje de programación y programar.
- **Pruebas**: detección de errores y depuración.
- **Documentación**: documentar todas la etapas.
- **Explotación**: instalación, configuración y prueba en el equipo del cliente (en producción).
- **Mantenimiento**: actualización y depuración.



Las pruebas de software consisten en ***verificar*** y ***validar*** un producto software antes de su puesta en marcha.

- La **verificación** es la comprobación de que un sistema o parte de un sistema, cumple con las condiciones impuestas. Con la verificación se comprueba si la aplicación se está construyendo *correctamente.*

¿Estamos construyendo el producto correctamente?

- La **validación** es el proceso de evaluación del sistema o de uno de sus componentes, para determinar si satisface *los requisitos especificados*.

¿Estamos construyendo el producto correcto?

Pruebas

- Representan una revisión final de las especificaciones, del diseño y de la codificación.
- Objetivo: Descubrir un error.
  El sistema tenga un comportamiento determinado, que la interfaz cumpla una apariencia y unas características determinadas.
- Un caso de prueba es bueno cuando su ejecución conlleva una probabilidad elevada de encontrar un error y tiene éxito cuando lo detecta.

Técnicas de Diseño de Casos de Prueba

Un **caso de prueba** es un conjunto de entradas, condiciones de ejecución y resultados esperados, desarrollados para conseguir un objetivo particular o condición de prueba.

1. Definir precondiciones y postcondiciones.
2. Identificar unos valores de entrada y establecer su comportamiento.
3. Comprobar si ese comportamiento es el previsto o no.

Compromiso entre cantidad de recursos y probabilidad de éxito.

Principios básicos de las pruebas

International Software Testing Qualifications Board
https://www.istqb.org/

https://www.istqb.org/downloads/send/2-foundation-level-documents/281-istqb-ctfl-syllabus-2018-v3-1.html

https://youtu.be/rFaWOw8bIMM

Principios básicos de las pruebas

Principio1:

Las pruebas demuestran la presencia de errores, pero no su ausencia.

Principios básicos de las pruebas

Principio2:

Las pruebas completas no existen

Principios básicos de las pruebas

Principio3:

Las pruebas se iniciarán lo antes posible en el ciclo de vida de software

Principios básicos de las pruebas

Principio4:

La mayor parte de los errores se suele concentrar en un número reducido de módulos.

Principios básicos de las pruebas

Principio5:

Si las pruebas se repiten una y otra vez, con el tiempo el mismo conjunto de pruebas ya no encontrará nuevos errores. Los casos de prueba deben ser examinados, revisados y actualizados periódicamente

Principios básicos de las pruebas

Principio6:

Las pruebas deben adaptarse a la necesidades específicas del contexto

Principios básicos de las pruebas

Principio7:

Un software puede haber pasado todas las fases de pruebas, pero esto no indica que no pueda tener errores que aún no se han logrado identificar.

Enfoque Funcional o Caja Negra

Enfoque Estructural o Caja Blanca

Pruebas Funcionales

Evalúa lo que hace el sistema, mas que cómo lo hace.

Tipos de Prueba:

1.- Particiones Equivalentes.
2.- Análisis de Valores Límite.
3.- Pruebas Aleatorias.

Pruebas Estructurales

Evalúa de forma detallada la arquitectura de la aplicación, cómo se hace.

Se basa en unos criterios de cobertura lógica, cuyo cumplimiento determina la mayor o menor seguridad en la detección de errores.

Se plantean distintos caminos de ejecución alternativos y se llevan a cabo para observar los resultados y contrastarlos con lo esperado.

Tipos de Prueba:

Pruebas de Cobertura: Sentencias, decisiones, camino básico….

Pruebas de Regresión

Estas pruebas intentan descubrir si existe algún error tras la modificaciones o si puede haber algún problema no detectado hasta ahora.

Las pruebas de regresión no sirven no tienen la finalidad de detectar nuevos errores, sino para certificar su ausencia, que no aparezcan errores durante las pruebas no significa que el software esté exento de ellas.

Estrategia de pruebas

Documentación de Pruebas

Los documentos que se van a generar son:

- **Plan de Pruebas**: Al principio se desarrollará una planificación general, que quedará reflejada en el "Plan de Pruebas". El plan de pruebas se inicia el proceso de Análisis del Sistema.
- **Especificación del diseño de pruebas.** De la ampliación y detalle del plan de pruebas, surge el documento "Especificación del diseño de pruebas".
- **Especificación de un caso de prueba**. Los casos de prueba se concretan a partir de la especificación del diseño de pruebas.
- **Especificación de procedimiento de prueba**. Una vez especificado un caso de prueba, será preciso detallar el modo en que van a ser ejecutados cada uno de los casos de prueba, siendo recogido en el documento "Especificación del procedimiento de prueba".
- **Registro de pruebas**. En el "Registro de pruebas" se registrarán los sucesos que tengan lugar durante las pruebas.
- **Informe de incidente de pruebas**. Para cada incidente, defecto detectado, solicitud de mejora, etc, se elaborará un "informe de incidente de pruebas".
- **Informe sumario de pruebas**. Finalmente un "Informe sumario de pruebas" resumirá las actividades de prueba vinculadas a uno o más especificaciones de diseño de pruebas.

Documentación de Pruebas

https://cmmiinstitute.com/

https://conasa.grupocibernos.com/blog/la-importancia-del-cmmi-en-el-desarrollo-de-software

Pruebas de Código

1.- Prueba del Camino Básico.
2.- Clases de Equivalencia.
3.- Análisis de Valores límite.

Pruebas Caja Blanca

- Seleccionar un conjunto de caminos lógicos y generar datos de prueba, determinando valores específicos que definen la ejecución de esos caminos seleccionados.
- Estos casos deben garantizar:
  - Que se ejercita por lo menos una vez todos los caminos independientes de cada módulo: **prueba de cobertura de sentencias.**
  - Que se ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa: **prueba de cobertura de condición.**
  - Que se ejecuten todos los bucles en sus límites operacionales: **prueba de bucles**.
  - Que se ejerciten las estructuras internas de datos para asegurar su validez.

"Los errores se esconden en los rincones y se aglomeran en los límites" [Beizer].

Pruebas del Camino Básico

- Es una técnica para obtener casos de prueba de **caja blanca**.
- Este método permite obtener una medida de la complejidad lógica de un diseño procedimental.
- Esta medida puede ser usada como guía a la hora de **definir un conjunto básico de caminos de ejecución** (diseño de casos de prueba).
- Para la obtención de la complejidad lógica o ciclomática emplearemos una representación del flujo de control en forma de grafo (**Grafo del flujo**).

Grafos de Flujo

Representan el flujo de control del programa

Están compuestos por:

- Nodos
  - Bloques de código (una o más sentencias)
- Arcos/aristas
  - Representan el flujo de control
  - Son las conexiones entre los bloques de código
- Regiones
  - Áreas delimitadas por aristas y nodos
- Nodos predicado
  - Nodo que contiene una condición
  - Se caracteriza porque de él salen dos o más aristas

Una de las características para comprobar si un grafo de flujo está bien diseñado es que los únicos nodos de los que pueden partir dos aristas son los nodos predicado

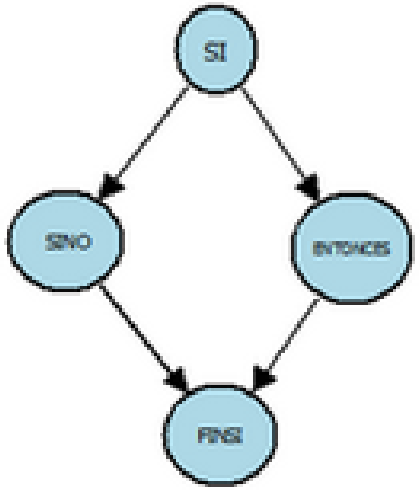
Grafos de Flujo: Estructuras

SECUENCIAL

Instrucción 1
Instrucción 2
...
Instrucción n

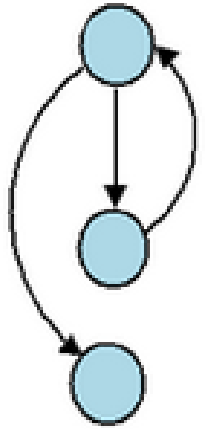
CONDICIONAL

SI <condición> ENTONCES
    <Instrucciones>
SINO
    <Instrucciones>
FIN SI



HACER MIENTRAS

MIENTRAS <condicion> HACER
    <Instrucciones>
FIN MIENTRAS

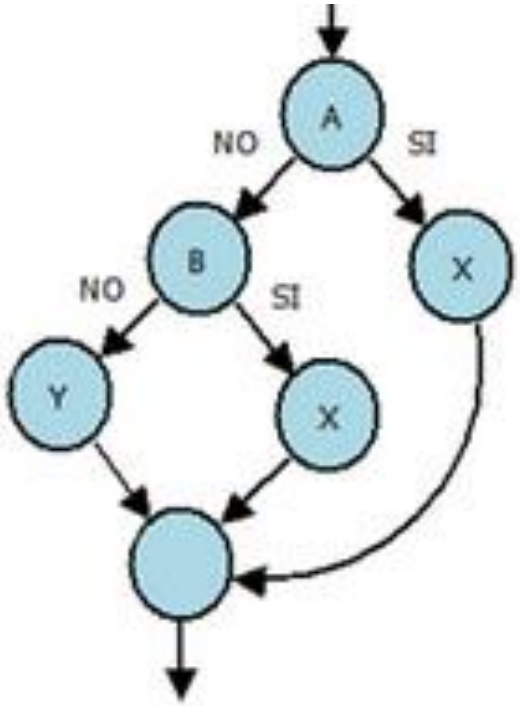


REPETIR HASTA

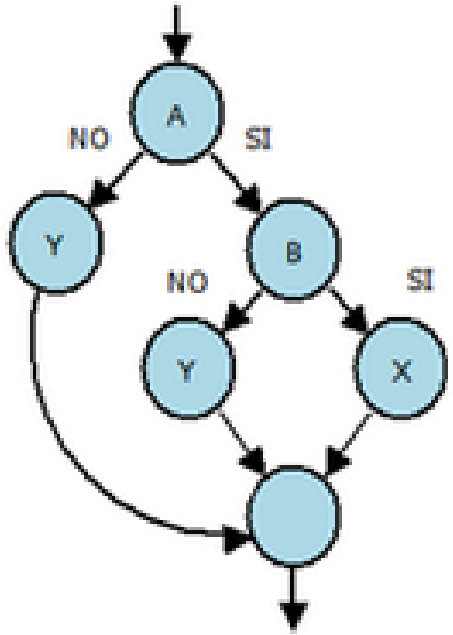
REPETIR
    <Instrucciones>
HASTA QUE <condicion>

Grafos de Flujo: Estructuras de Control

```
SI A OR B ENTONCES
    PROC X
SINO
    PROC Y
FIN SI
```

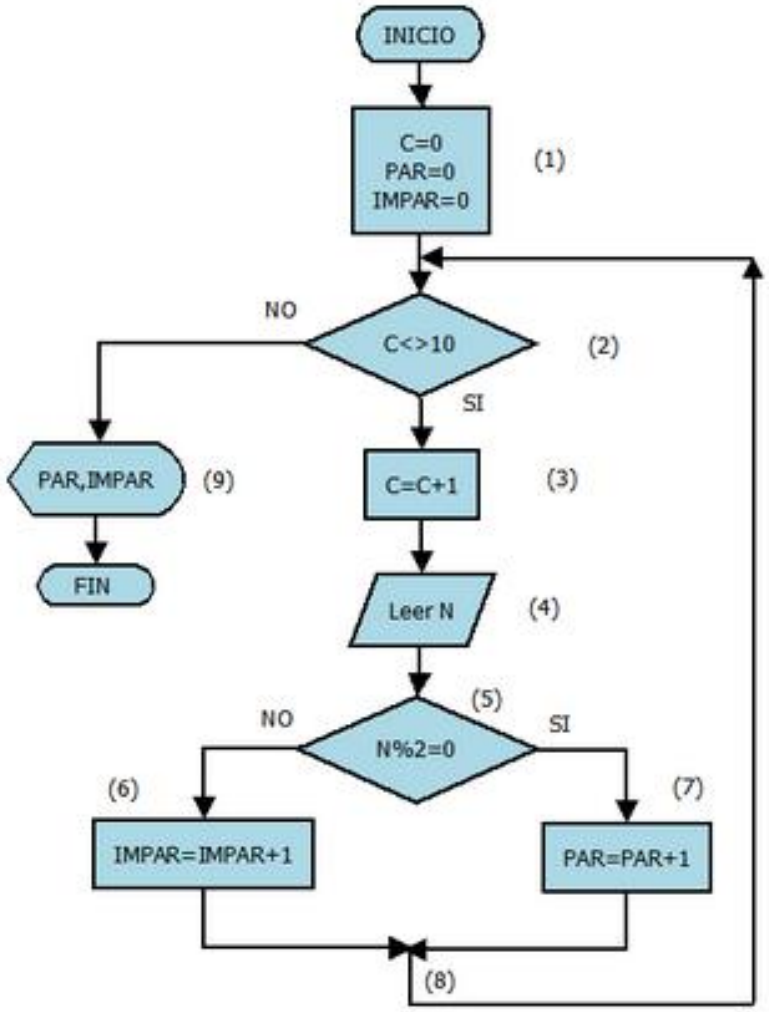


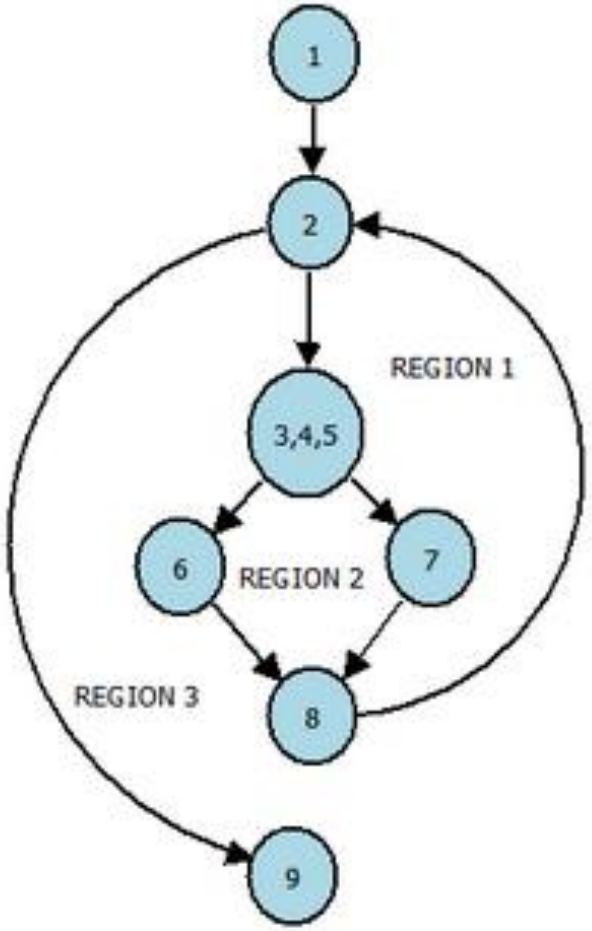

```
SI A AND B ENTONCES
    PROC X
SINO
    PROC Y
FIN SI
```



Ejemplo1

Partimos del diagrama de flujo de un programa que lee 10 números por teclado y muestra cuántos de los números son pares y cuantos impares. La estructura principal corresponde a un Mientras y dentro hay una estructura IF.





Cálculo de la complejidad ciclomática

- Es una **medida del software** que aporta una valoración cuantitativa de la complejidad lógica de un programa.
- Define el **número de caminos independientes** de un programa.
- Por **camino independiente** se entiende aquel que introduce un nuevo conjunto de sentencias o una nueva condición. En términos del grafo, por una arista que no haya sido recorrida antes.
- Nos da una **cota o límite superior** para el número de casos de prueba.

Cálculo de la complejidad ciclomática

Formas de cálculo:

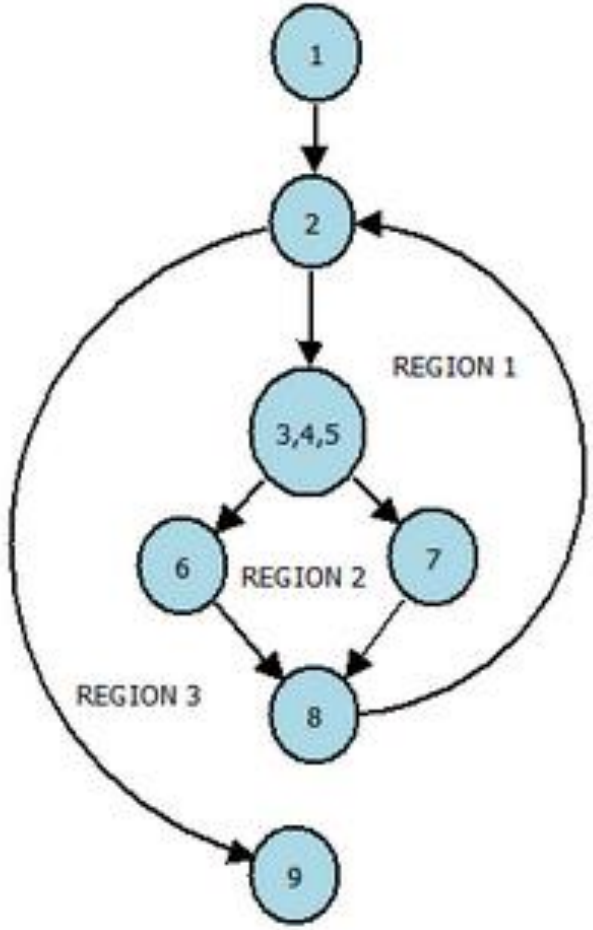
1. El número de **regiones** del grafo.
2. **V(G) = A – N + 2**

   Donde A es el número de aristas y N es el número de nodos contenidos en el grafo.
3. **V(G) = P + 1**

   Donde P es el número de nodos predicados contenidos en el grafo

Ejemplo1

Partimos del diagrama de flujo de un programa que lee 10 números por teclado y muestra cuántos de los números son pares y cuantos impares. La estructura principal corresponde a un Mientras y dentro hay una estructura IF.



$$V(G) = 3$$

Número de regiones =3

Aristas - nodos + 2 = 8 - 7 + 2 = 3

Nodos predicado + 1 = 2 + 1 = 3

Camino 1: 1-2-9

Camino 2: 1-2-3,4,5-6-8-2-9

Camino 3: 1-2-3,4,5-7-8-2-9

Obtención de los casos de prueba

El último paso de la prueba del camino básico es construir los casos de prueba que fuerzan la ejecución de cada camino. Con el fin de comprobar cada camino, debemos escoger los casos de prueba de forma que las condiciones de los nodos predicado estén adecuadamente establecidas. Una forma de representar el conjunto de casos de prueba es como se muestra en la siguiente tabla.

Ejemplo1

Camino	Caso de prueba	Resultado esperado
1	Escoger algún valor de C tal que NO se cumpla la condición C<>10 C=10	Visualizar le número de pares y el de impares
2	Escoger algún valor de C tal que SI se cumpla la condición C<>10 Escoger algún valor de N tal que NO se cumpla la condición N%2=0 C=1, N=5	Contar números impares
3	Escoger algún valor de C tal que SI se cumpla la condición C<>10 Escoger algún valor de N tal que SÍ se cumpla la condición N%2=0 C=2, N=4	Contar números pares

Ejemplo2

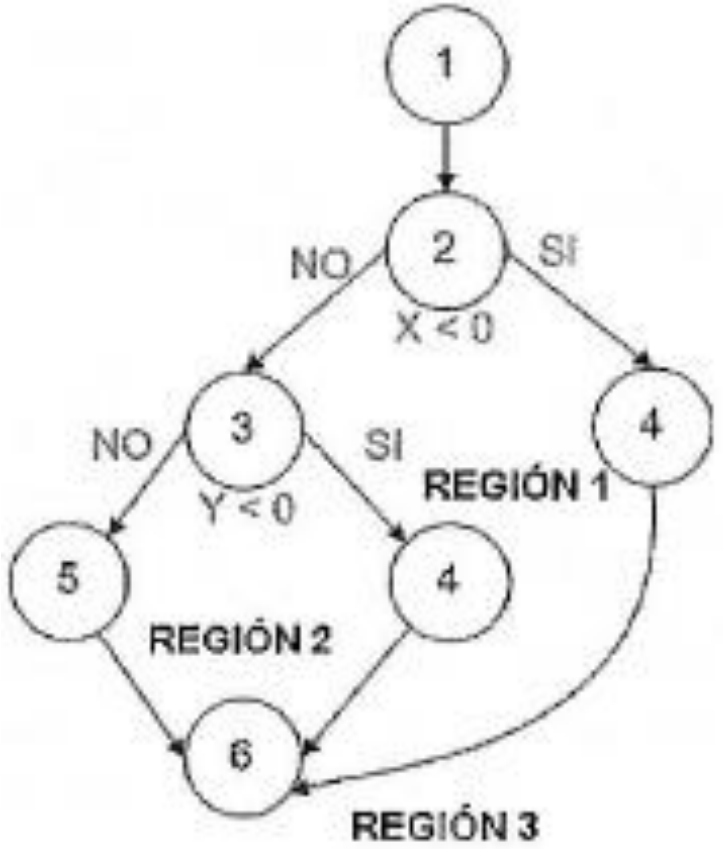
Dado la siguiente función Java que calcula la media de dos números, dibuja el grafo de flujo y calcula la complejidad ciclomática y los caminos independientes. Establece los casos de prueba para cada camino.

```
static void visualizarMedia(float x, float y){
  float resultado=0;
  if (x<0 || y<0){
    System.out.println(" x e y deben ser positivos");
  } else {
      resultado = (x+y)/2;
      System.out.println("La media es: "+ resultado);
    }

}
```

Ejemplo2

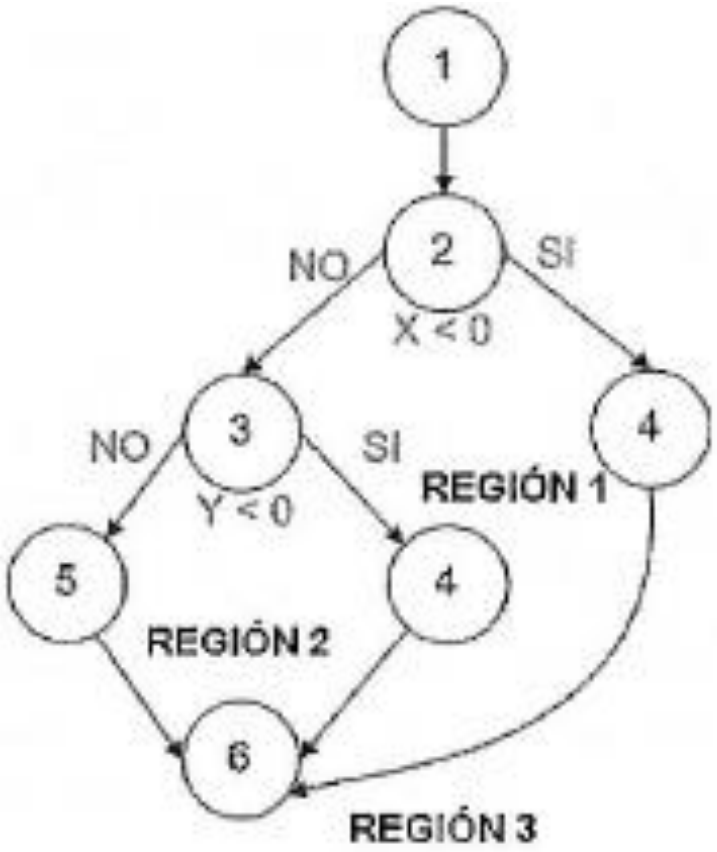
Dado la siguiente función Java que calcula la media de dos números, dibuja el grafo de flujo y calcula la complejidad ciclomática y los caminos independientes. Establece los casos de prueba para cada camino.

```
static void visualizarMedia(float x, float y){
   float resultado=0;
   if (x<0 || y<0){
      System.out.println(" x e y deben ser positivos");
   } else {
       resultado = (x+y)/2;
       System.out.println("La media es: "+ resultado);
     }

}
```



Ejemplo2

Dado la siguiente función Java que calcula la media de dos números, dibuja el grafo de flujo y calcula la complejidad ciclomática y los caminos independientes. Establece los casos de prueba para cada camino.

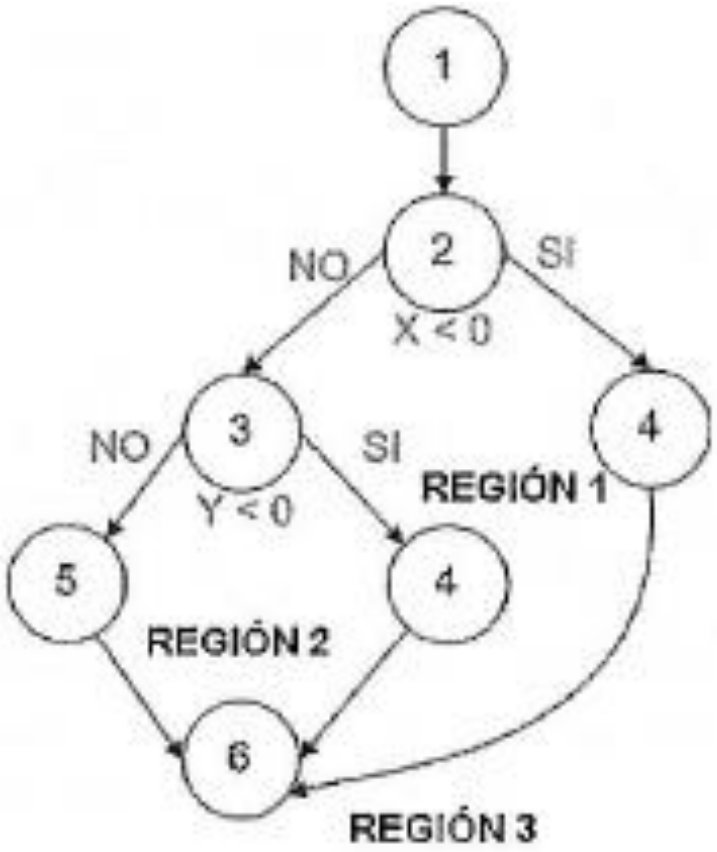


1.- V(G) = Número de regiones = 3
2.- V(G) = Aristas – Nodos +2 = 8 – 7 +2 = 3
3.- V(G) = Nodos Predicado +1 = 3

Camino 1: 1-2-3-5-6
Camino 2: 1-2-4-6
Camino 3: 1-2-3-4-6

Ejemplo2

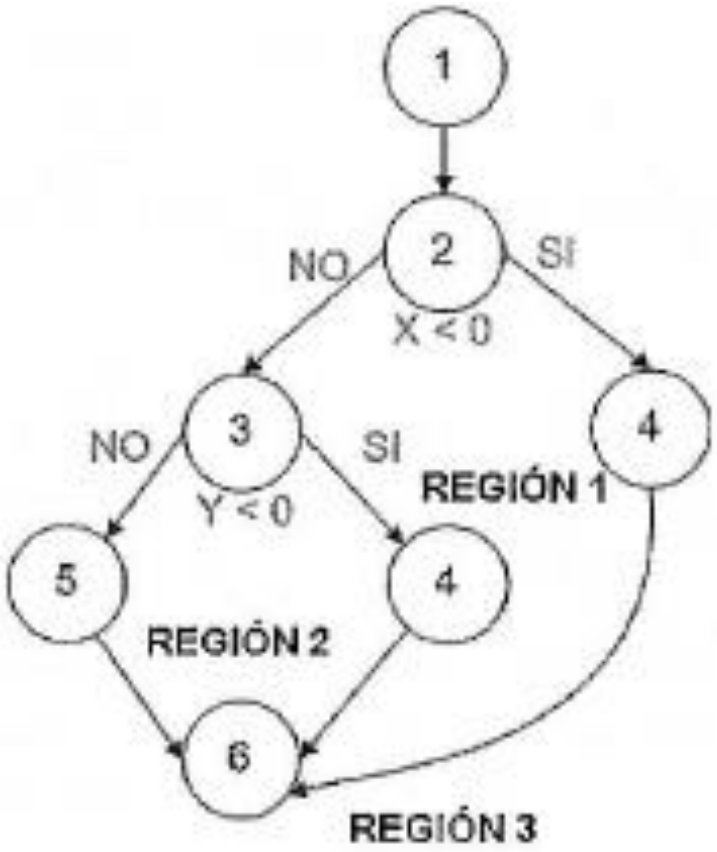
Dado la siguiente función Java que calcula la media de dos números, dibuja el grafo de flujo y calcula la complejidad ciclomática y los caminos independientes. Establece los casos de prueba para cada camino.



Camino	Caso de prueba	Resultado esperado
Camino 1: 1-2-3-5-6	Escoger algún X e Y tal que NO se cumpla la condición X<0 || Y<0 X=4, Y=5 visualizarMedia(4,5)	Visualiza: La media es: 4.5
Camino 2: 1-2-4-6	Escoger algún X tal que SÍ se cumpla la condición X<0 (Y puede ser cualquier valor) X=-4, Y=5 visualizarMedia(-4,5)	Visualiza: X e Y deben ser positivos
Camino 3: 1-2-3-4-6	Escoger algún X tal que NO cumpla la condición X<0 y escoger algún Y que SÍ cumpla la condición Y<0 X=4, Y=-5 visualizarMedia(4,-5)	Visualiza: X e Y deben ser positivos

Ejemplo2

Dado la siguiente func
de flujo y calcula la co
los casos de prueba pa



Camino	Caso de prueba	Resultado esperado
Camino 1: 1-2-3-5-6	Escoger algún X e Y tal que NO se cumpla la condición X<0 || Y<0 X=4, Y=5 visualizarMedia(4,5)	Visualiza: La media es: 4.5
Camino 2: 1-2-4-6	Escoger algún X tal que SÍ se cumpla la condición X<0 (Y puede ser cualquier valor) X=-4, Y=5 visualizarMedia(-4,5)	Visualiza: X e Y deben ser positivos
Camino 3: 1-2-3-4-6	Escoger algún X tal que NO cumpla la condición X<0 y escoger algún Y que SÍ cumpla la condición Y<0 X=4, Y=-5 visualizarMedia(4,-5)	Visualiza: X e Y deben ser positivos

```
static void visualizarMedia(float x, float y){
  float resultado=0; 1
  if (x<0 || y<0){ 2 3
    System.out.println(" x e y deben ser positivos"); 4
  } else {
      resultado = (x+y)/2;
      System.out.println("La media es: "+ resultado); 5
    }
  6
}
```

Pruebas Funcionales

Tipos de errores que detecta:

- Funciones incorrectas o ausentes
- Errores de la interfaz
- Errores en estructuras de datos o accesos a bases de datos
- Errores de rendimiento
- Errores de inicialización y terminación

Clases de Equivalencia

- Es una técnica para obtener casos de prueba de **caja negra**.
- Se examina una condición de entrada y hay que cubrir todo el rango de valores posible que tenga:

Entradas Válidas	Clases de Equivalencia Válidas
Entradas No Válidas	Clases de Equivalencia NO Válidas

1. Si una condición de entrada especifica un **rango de valores** (ej: valor entre 1 y 99)
   - se definen una clase de equivalencia válida: *1<= valor <= 99*
   - y dos inválidas: *valor <1* y *valor > 99*
2. Si una condición de entrada requiere un **valor específico**, (ej. El tamaño de un código es de 5 dígitos), se definen
   - una clase de equivalencia válida: *tamaño del código 5,*
   - y dos inválidas: *tamaño < 5 y tamaño > 5*
3. Si una condición de entrada especifica un **conjunto de valores** de entrada: (ej. *Tipo de vehícul*
   AUTOBÚS, TAXI, TURISMO; se define
   - una clase de equivalencia válida para cada valor
   - y una inválida, será un valor distinto a los posibles (por ejemplo *PATINES*).
4. Si una condición de entrada es **booleana**, (ej. *El primer carácter de un identificador debe ser una letra)*, S
   definen
   - una clase válida: *es letra,*
   - y una inválida: *no es letra*

Condiciones de entrada	Nº de Clases de equivalencia válidas	Nº de Clases de equivalencia no válidas
1 . Rango	1 CLASE VÁLIDA Contempla los valores del rango	2 CLASES NO VÁLIDAS Un valor por encima del rango Un valor por debajo del rango
2. Valor específico	1 CLASE VÁLIDA Contempla dicho valor	2 CLASES NO VÁLIDAS Un valor por encima Un valor por debajo
3. Miembro de un conjunto	1 CLASE VÁLIDA Una clase por cada uno de los miembros del conjunto	1 CLASE NO VÁLIDA Un valor que no pertenece al conjunto
4. Lógica	1 CLASE VÁLIDA Una clase que cumpla la condición	1 CLASE NO VÁLIDA Una clase que no cumpla la condición

Clases de Equivalencia: Tabla

Las clases identificadas se consignan en una tabla, enumerándolas de cara a la derivación de los casos de prueba

Condición de Entrada	Clases de Equivalencia	Clases Válidas	COD	Clases No Válidas	COD

Clases de Equivalencia: Determinar Casos de Prueba

Hay que identificar los casos de prueba para cada clase de equivalencia.

Pasos:

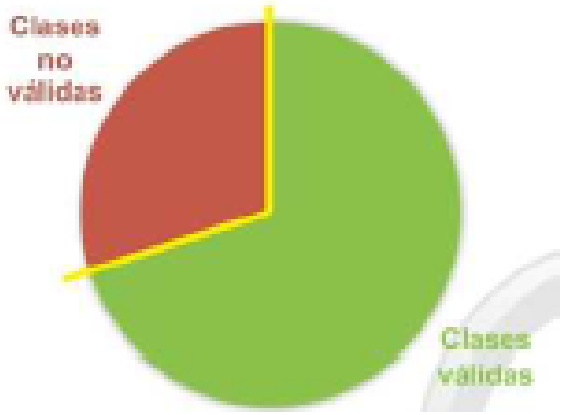
1. Asignar un **número único** a cada clase de equivalencia.
2. Hasta que todas las clases de equivalencia **válidas** hayan sido cubiertas por casos de prueba, escribir **un nuevo caso de prueba** que cumpla **tantas clases de equivalencia válidas no cubiertas como sea posible.**
3. Hasta que todas las clases de equivalencia **inválidas** hayan sido cubiertas por casos de prueba, escribir **un caso de prueba** para cubrir **una y solo una** de las clases de equivalencia no cubierta.

Clases de Equivalencia: Determinar Casos de Prueba

CASO DE PRUEBA	Clases de Equivalencia	CONDICIONES DE ENTRADA	Resultado Esperado
CP1			
CP2			
CP3			
CP4			
CP5			
CP6			

Análisis de los Valores Límite

- Se basa en la evidencia experimental de que los errores suelen aparecer con mayor probabilidad en **los extremos** de los campos de entrada.
- Un análisis de las condiciones límite de las clases de equivalencia aumenta la eficiencia de la prueba
    - **Condiciones límite**: valores justo por encima y por debajo de los márgenes de la clase de equivalencia.



Análisis de los Valores Límite: Cálculo

Condiciones de la especificación	Obtención de los casos de prueba
1. Rango de valores como condición de entrada	1 caso que ejercite el valor máximo del rango
	1 caso que ejercite el valor mínimo del rango
	1 caso que ejercite el valor justo por encima del máximo del rango
	1 caso que ejercite el valor justo por debajo del mínimo del rango
2. Valor numérico específico como condición de entrada	1 caso que ejercite el valor numérico específico
	1 caso que ejercite el valor justo por encima del valor numérico específico
	1 caso que ejercite el valor justo por debajo del valor numérico específico
3. Rango de valores como condición de salida	Generar casos de prueba según el criterio 1 que ejerciten dichas condiciones de salida
4. Valor numérico específico como condición de salida	Generar casos de prueba según el criterio 2 que ejerciten dichas condiciones de salida
5. Estructura de datos como condición de salida o de entrada	1 caso que ejercite el primer elemento de la estructura
	1 caso que ejercite el último elemento de la estructura

Análisis de los Valores Límite: Ejemplo

	Condiciones de entrada y salida	Casos de prueba
Código	Entero de 1 a 100	Valores: 0, 1, 100, 101
Puesto	Alfanumérico de hasta 4 caracteres	Longitud de caracteres: 0, 1, 4, 5
Antigüedad	De 0 a 25 años (Real)	Valores: 0, 25, -0.1, 25.1
Horas semanales	De 0 a 60	Valores: 0, 60, -1, 61
Fichero de entrada	Tiene de 1 a 100 registros	Para leer 0, 1, 100 y 101 registros
Fichero de salida	Podrá tener de 0 a 10 registros	Para generar 0, 10 y 11 registros (no se puede generar -1 registro)
Array interno	De 20 cadenas de caracteres	Para el primer y último elemento.

Ejemplo1

Tenemos una función Java que recibe un número entero y devuelve una cadena de caracteres con el texto "PAR" si el número recibido es par o "IMPAR" si el número recibido es impar.

```
public String parImpar(int numero) {
    String cadena = "";
    if (numero % 2==0) {
        cadena="PAR";
    } else
        cadena="IMPAR";
    return cadena;
```

Ejemplo1

Tenemos una función Java que recibe un número entero y devuelve una cadena de caracteres con el texto "PAR" si el número recibido es par o "IMPAR" si el número recibido es impar.

```
public String parImpar(int numero) {
     String cadena = "";
     if (numero % 2==0) {
          cadena="PAR";
     } else
          cadena="IMPAR";
     return cadena;
```

Condición de Entrada	Clases de Equivalencia	Clases Válidas	COD	Clases No Válidas	COD
numero	Valor Par	Cualquier Numero Par	V1 o 1	Numero Impar Cadena	NV1 o 3 NV2 o 4
	Valor Impar	Cualquier Numero Impar	V2 o 2	Numero par cadena	NV3 o 5 NV4 o 6

Ejemplo1

Tenemos una función Java que recibe un número entero y devuelve una cadena de caracteres con el texto "PAR" si el número recibido es par o "IMPAR" si el número recibido es impar.

```
public String parImpar(int numero) {
     String cadena = "";
     if (numero % 2==0) {
          cadena="PAR";
     } else
          cadena="IMPAR";
     return cadena;
```

Condición de Entrada	Clases de Equivalencia	Clases Válidas	COD	Clases No Válidas	COD
numero	Valor Par	Cualquier Numero Par	V1 o 1	Numero Impar Cadena	NV1 o 3 NV2 o 4
	Valor Impar	Cualquier Numero Impar	V2 o 2	Numero par cadena	NV3 o 5 NV4 o 6

CASO DE PRUEBA	Clases de Equivalencia	CONDICIONES DE ENTRADA	Resultado Esperado
		numero	
CP1	V1 o 1	20	PAR
CP2	V2 o 2	25	IMPAR
CP3	NV1 o 3	45	Error número impar
CP4	NV2 o 4	"hola"	Error cadena
CP5	NV3 o 5	10	Error número par
CP6	NV4 o 6	"adiós"	Error cadena