DISEÑO Y REALIZACIÓN DE PRUEBAS

RA3 - Entornos de Desarrollo

Pruebas

“La prueba de software puede ser usada para mostrar la presencia de bugs, pero nunca su ausencia” Dijkstra (1970)

“Es una actividad realizada para evaluar la calidad del producto y mejorarla, identificando defectos y problemas” Swebook

Prueba de software: “Es la verificación dinámica del comportamiento de un programa contra el comportamiento esperado, usando un conjunto finito de casos de prueba, seleccionados de manera adecuada”

En este capítulo aprenderemos a utilizar diferentes técnicas para elaborar casos de prueba. Usaremos una herramienta de depuración definiendo puntos de ruptura y examinando variables durante la ejecución de un programa. Aprenderemos a utilizar la herramienta JUNIT para elaborar pruebas unitarias para clases Java.

PRUEBA DE PROGRAMAS

Cuando se habla de pruebas del software o de programas es común utilizar los términos “prueba” y “caso de prueba”. Su definición es la siguiente:

- **Prueba**: es el proceso de ejecución de un programa con el intento deliberado de encontrar errores. (Caja negra, prueba funcional)
- **Caso de prueba**: conjunto de entradas, condiciones de ejecución y resultados esperados diseñados para un objetivo particular de condición de prueba. (Caja blanca, prueba estructural)

PRINCIPIOS BÁSICOS DE LAS PRUEBAS

Según el ISTQB (International Software Testing Qualifications Board) los principios básicos que guían las pruebas del software son:

- Principio 1: Las pruebas demuestran la presencia de errores, pero no su ausencia.
- Principio 2: Las pruebas completas no existen
- Principio 3: Las pruebas se iniciarán lo antes posible en el ciclo de vida del software.
- Principio 4: La mayor parte de los errores se suelen concentrar en un número reducido de módulos.
- Principio 5: Si las pruebas se repiten una y otra vez, con el tiempo el mismo conjunto de casos de prueba ya no encontrará nuevos errores. Los casos de prueba deben ser examinados y revisados periódicamente.
- Principio 6: Las pruebas deben adaptarse a las necesidades específicas del contexto.
- Principio 7: 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.

Validación y verificación

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

Ambos procesos son esenciales para garantizar la calidad del software. Mientras que la verificación se asegura de que el software se esté construyendo correctamente, la validación se asegura de que se esté construyendo el software correcto.

Validación

Proceso de evaluación de un sistema o componente durante o al final del proceso de desarrollo para determinar cuándo se satisfacen los requerimientos especificados

¿Se construyó el producto correcto?

Implica la ejecución del código y se centra en comprobar si el producto final cumple con los requisitos y expectativas del usuario.

La validación se lleva a cabo en etapas más avanzadas del proyecto, ya que requiere un producto funcional o código que ejecutar.

Incluye actividades como pruebas unitarias, pruebas de integración, pruebas del sistema y pruebas de aceptación del usuario.

Verificación:

Proceso de evaluación de un sistema o componente para determinar si un producto de una determinada fase de desarrollo satisface las condiciones impuestas al inicio de la fase

¿Se está construyendo el producto de acuerdo a los requisitos y especificaciones?

Se realiza en varias fases del desarrollo del software, desde la revisión de documentos y diseño hasta la inspección del código.

Al hacer esto, la verificación ayuda a identificar errores y defectos en las primeras etapas del ciclo de desarrollo, lo cual ahorra tiempo y costes al reducir la probabilidad de implementar incorrectamente las especificaciones.

Diferencias Clave

1. **Enfoque y Momento**: La verificación se realiza antes de la validación y se centra en el proceso de desarrollo, mientras que la validación ocurre después y se enfoca en el producto terminado.
2. **Métodos Utilizados**: La verificación emplea revisiones y análisis de documentación, mientras que la validación implica pruebas de software, como pruebas de caja negra y blanca, y pruebas no funcionales.
3. **Equipos Involucrados**: Generalmente, la verificación es llevada a cabo por el equipo de aseguramiento de calidad (QA), mientras que la validación implica al equipo de pruebas en colaboración con QA.
4. **Objetivos**: La verificación busca confirmar que el software cumple con las especificaciones, mientras que la validación verifica si el software satisface las necesidades y expectativas del usuario.

Ejemplo: Desarrollo de una Aplicación de Compras en Línea

Fase de Verificación

Imagina que estás desarrollando una aplicación de compras en línea. Durante la fase de verificación, el equipo de aseguramiento de calidad (QA) revisa el diseño y el código fuente para asegurarse de que se están siguiendo las especificaciones y requisitos del cliente. Por ejemplo, verifican que la aplicación tenga una función de búsqueda y un carrito de compras, tal como se especificó en los documentos de diseño.

En este contexto, la verificación podría incluir:

- **Revisión de Documentación**: Asegurarse de que las especificaciones del cliente estén correctamente documentadas.
- **Inspección de Código**: Verificar que el código fuente refleje las funcionalidades requeridas, como una interfaz de usuario limpia para la búsqueda de productos.
- **Revisión de Diseño**: Chequear que el diseño de la base de datos soporte todas las operaciones de la aplicación, como añadir o eliminar productos del carrito.

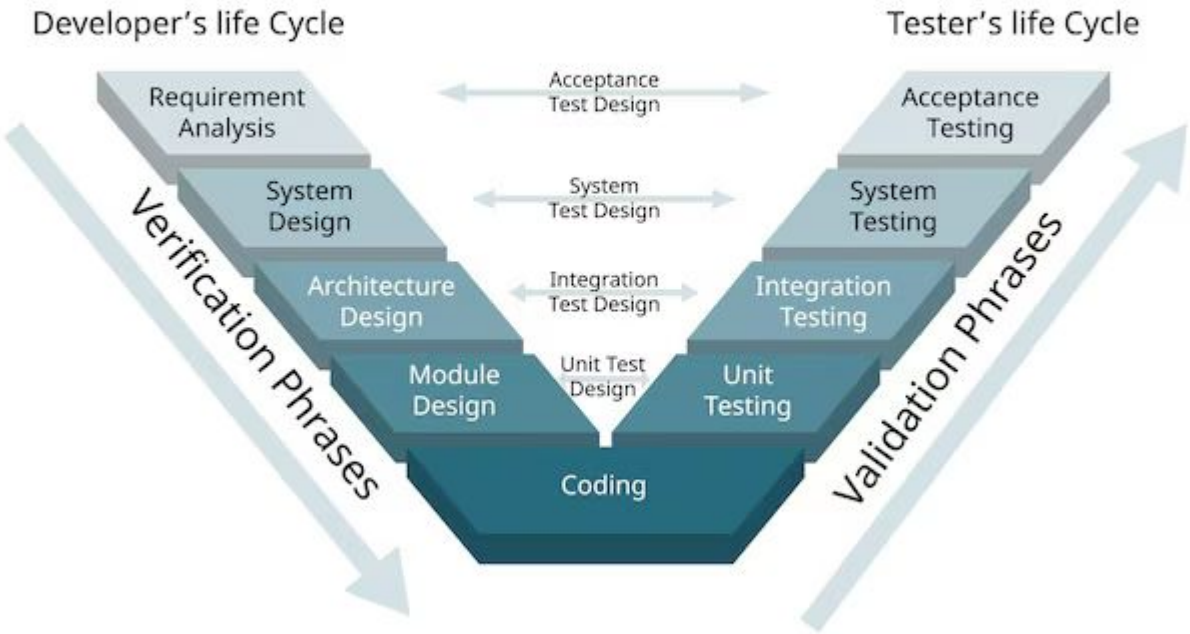
Ejemplo: Desarrollo de una Aplicación de Compras en Línea

Fase de Validación

Una vez que la aplicación se desarrolla y está en una fase funcional, comienza la validación. Aquí, el equipo de pruebas ejecuta la aplicación para asegurarse de que cumple con las necesidades del usuario final. Por ejemplo, realizan pruebas para ver si la función de búsqueda devuelve resultados correctos y si el proceso de pago en el carrito de compras funciona sin errores.

La validación podría incluir:

- **Pruebas Unitarias**: Verificar que cada componente individual, como la función de búsqueda, funcione correctamente.
- **Pruebas de Integración**: Comprobar cómo los diferentes componentes de la aplicación, como el carrito de compras y el sistema de pago, trabajan juntos.
- **Pruebas del Sistema**: Asegurarse de que la aplicación completa funcione según lo esperado en diferentes dispositivos y navegadores.
- **Pruebas de Aceptación del Usuario**: Confirmar que la aplicación satisface las necesidades y expectativas del cliente final.

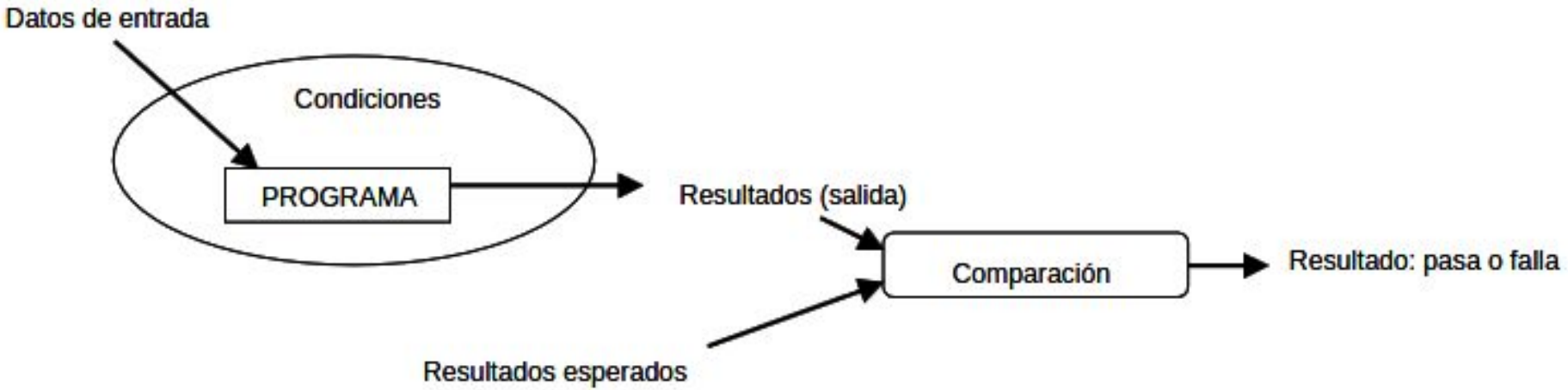

Developer's life Cycle
Tester's life Cycle
Requirement Analysis
System Design
Architecture Design
Module Design
Coding
Unit Testing
Integration Testing
System Testing
Acceptance Testing
Acceptance Test Design
System Test Design
Integration Test Design
Unit Test Design
Verification Phrases
Validation Phrases

Técnicas de diseño de caso de pruebas

Un caso de prueba es un conjunto de entradas, condiciones de ejecución y resultados esperados, desarrollado para lograr un objetivo específico o para probar una condición particular dentro de un software. Esto implica que un caso de prueba es una herramienta estructurada y detallada utilizada en el proceso de pruebas de software, y se compone de:

1. **Entradas**: Datos o acciones que se aplicarán en el software para realizar la prueba. Estas pueden variar desde simples entradas de usuario hasta flujos de datos más complejos.
2. **Condiciones de Ejecución**: El entorno o estado del software en el que se ejecutará la prueba. Esto puede incluir configuraciones específicas del sistema, estados de la aplicación, o cualquier requisito previo necesario para realizar la prueba.
3. **Resultados Esperados**: La salida o comportamiento esperado del software tras aplicar las entradas en las condiciones dadas. Estos resultados son utilizados para determinar si el software funciona correctamente bajo esas condiciones específicas.

Técnicas de diseño de caso de pruebas



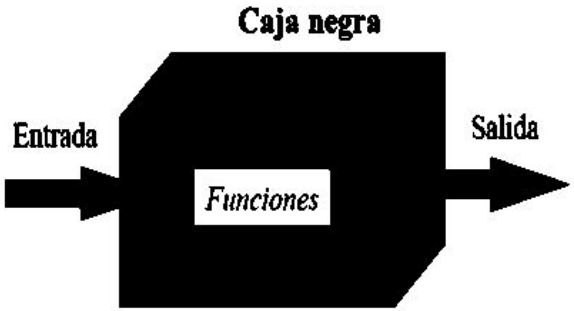
Técnicas de diseño de caso de pruebas

El objetivo de los **casos de prueba** es, por lo tanto, evaluar una funcionalidad específica o verificar que el software se comporte como se espera en determinadas condiciones.

Son fundamentales tanto en las pruebas de caja negra como en las de caja blanca, aunque el enfoque de su diseño y los detalles que contienen pueden variar significativamente entre estos dos métodos de prueba.

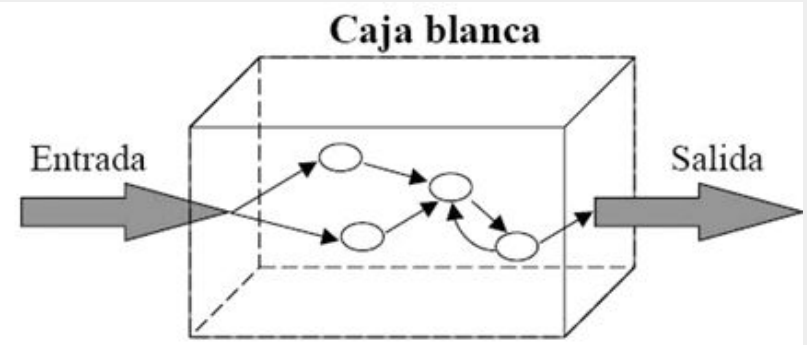
Las pruebas de caja negra examinan la funcionalidad del software sin conocer su estructura interna o código fuente. En este enfoque, el tester se centra en la entrada y salida del software, sin considerar cómo se procesan internamente estas entradas y salidas

Incluyen pruebas funcionales, pruebas no funcionales y pruebas de regresión

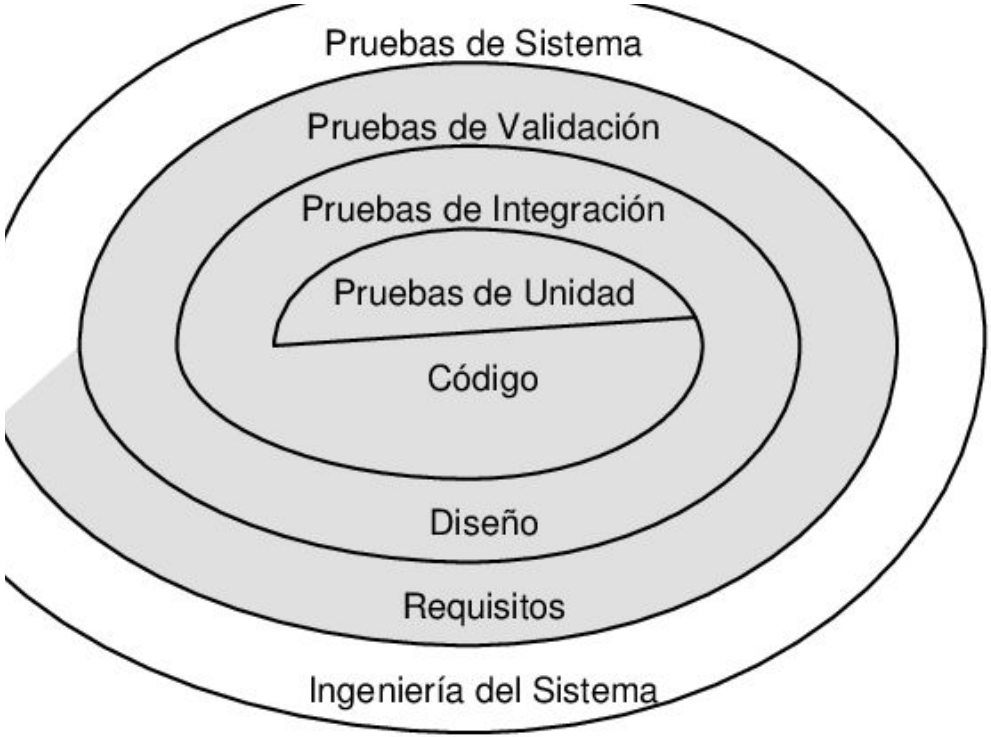


Las pruebas de caja blanca, también conocidas como pruebas de caja transparente o pruebas estructurales, se centran en la estructura interna del software. Los testers tienen conocimiento completo del código fuente y evalúan aspectos como la estructura de código, las ramificaciones, los caminos y las condiciones

Incluyen pruebas de cobertura de sentencias, pruebas de cobertura de ramas y pruebas de cobertura de caminos



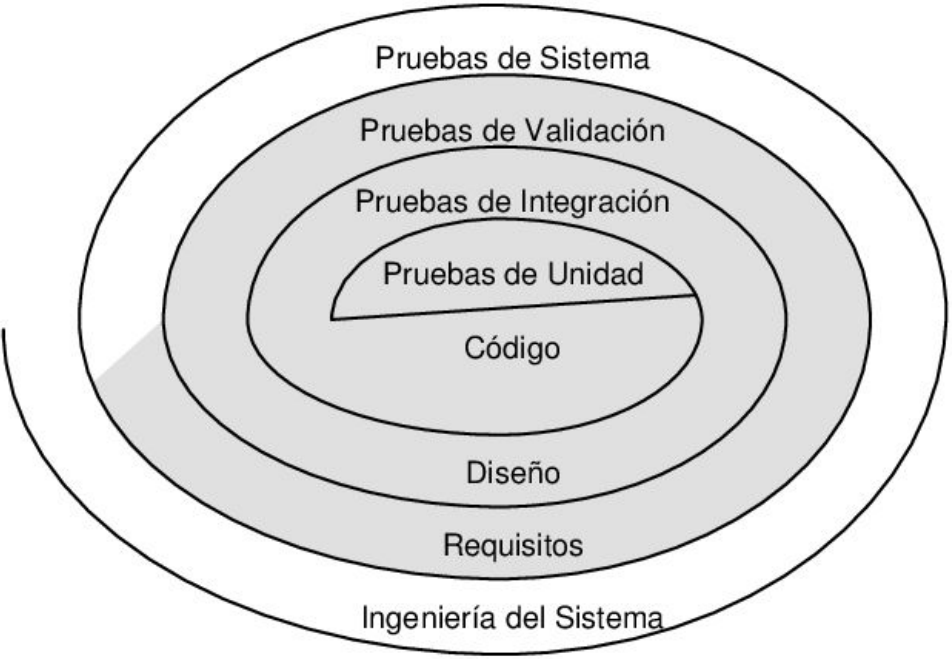
Estrategias de Pruebas del Software



Tipos de prueba:

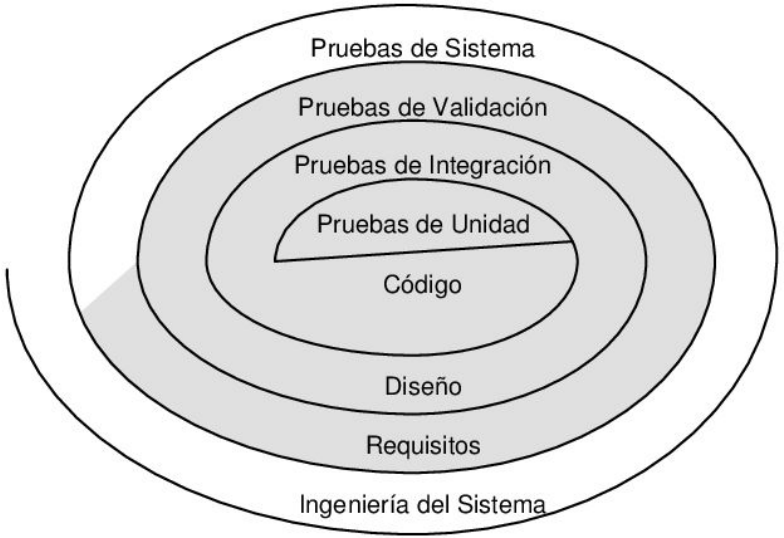
- En los módulos, Pruebas de unidad.
- En la unión de los módulos, Pruebas de integración.
- Cuando tenemos todos unidos, Prueba de validación.
- Cuando el sistema está funcionando, Prueba de sistema.

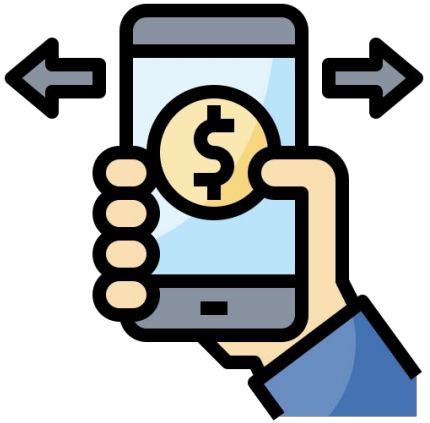
Estrategias de Pruebas del Software



- En el vértice de la espiral comienza la **prueba de unidad**. Se centra en la unidad más pequeña del software, el módulo tal como está implementado en código fuente
- La prueba avanza para llegar a la **prueba de integración.** Se toman los módulos probados mediante la prueba de unidad y se construye una estructura de programa que esté de acuerdo con lo que dicta el diseño. El foco de atención es el diseño
- La espiral avanza llegando a la **prueba de validación** (o de aceptación). Prueba del software en el entorno real de trabajo con intervención del usuario final. Se validan los requisitos establecidos como parte del análisis de requisitos del software comparándolos con el sistema que ha sido construido
- Finalmente se llega a la **prueba del sistema**. Verifica que cada elemento encaja de forma adecuada y se alcanza la funcionalidad y rendimiento total. Se prueba como un todo el software y otros elementos del sistema

Ejemplo
Estrategia de Pruebas





Imaginemos ahora el desarrollo y las pruebas de una aplicación móvil de banca en línea. Esta aplicación permitirá a los usuarios realizar operaciones bancarias como consultar saldos, transferir fondos y pagar facturas.

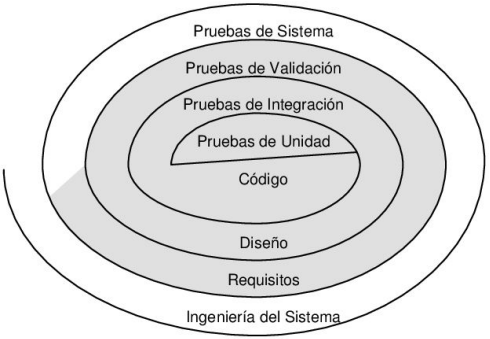
Ejemplo
Estrategia de Pruebas

Pruebas de Unidad

Pruebas de Integración

Pruebas de Validación (o de Aceptación)

Pruebas del Sistema



Pruebas de Unidad

Contexto: Los desarrolladores han creado módulos individuales como el inicio de sesión, consulta de saldo, transferencias y pagos de facturas.

Ejemplo Real: Un desarrollador escribe casos de prueba para validar la funcionalidad del inicio de sesión. Esto incluye pruebas para asegurar que el módulo maneje correctamente la autenticación y muestre mensajes de error adecuados para intentos de inicio de sesión fallidos.

Las **Pruebas de Unidad** se centran en la menor parte del software, como funciones o métodos, y aseguran que funcionen correctamente de manera aislada. Son fundamentales para identificar errores en el código en las etapas tempranas del desarrollo y suelen ser automatizadas para ejecutarse rápidamente

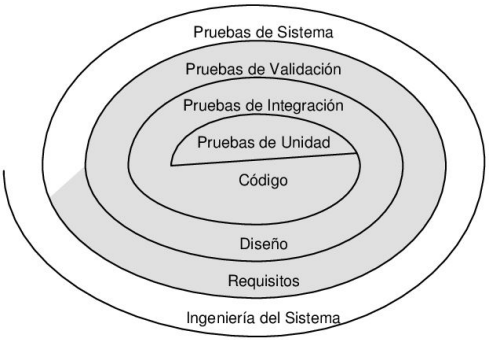
Ejemplo
Estrategia de Pruebas

Pruebas de Unidad

Pruebas de Integración

Pruebas de Validación (o de Aceptación)

Pruebas del Sistema



Pruebas de Integración

Contexto: Los módulos de inicio de sesión, consulta de saldo, transferencias y pagos de facturas deben trabajar juntos conforme al diseño del software.

Ejemplo Real: Se prueban flujos de trabajo donde, después de iniciar sesión con éxito, el usuario consulta su saldo y luego realiza una transferencia. Se verifica que la transición entre estas operaciones sea fluida y que la información del saldo se actualice correctamente después de la transferencia.

Las **Pruebas de Integración** evalúan cómo distintos módulos o componentes del software funcionan en conjunto.

- La integración no incremental, o "big bang", prueba todos los componentes simultáneamente,
- mientras que la integración incremental prueba módulos individualmente y luego en grupos, ya sea de forma ascendente o descendente, centrando la atención en el diseño y la arquitectura del sistema.

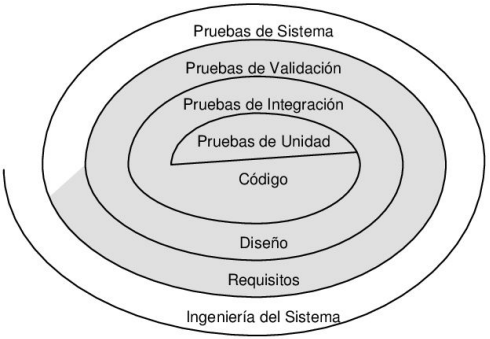
Ejemplo
Estrategia de Pruebas

Pruebas de Unidad

Pruebas de Integración

Pruebas de Validación (o de Aceptación)

Pruebas del Sistema



Pruebas de Validación (o de Aceptación)

Contexto: La aplicación se instala en dispositivos móviles y se invita a un grupo de usuarios reales a probarla en su entorno cotidiano.

Ejemplo Real: Un grupo de usuarios seleccionados utiliza la aplicación para llevar a cabo sus operaciones bancarias habituales. Se les pide que verifiquen si la aplicación satisface sus necesidades, como la facilidad de uso, la claridad de las instrucciones y la confiabilidad de las transacciones.

Las pruebas de validación incluyen
- la **Prueba Alfa**, que se realiza en presencia del equipo de desarrollo,
- y la **Prueba Beta**, que se lleva a cabo por un grupo de usuarios finales en un entorno de producción real. El objetivo es validar que el software cumple con los requisitos y es adecuado para su uso.

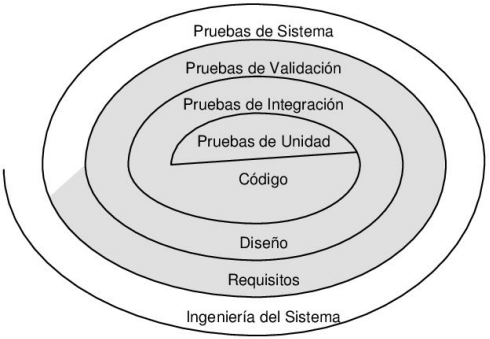
Ejemplo
Estrategia de Pruebas

Pruebas de Unidad

Pruebas de Integración

Pruebas de Validación (o de Aceptación)

Pruebas del Sistema



Pruebas del Sistema

Contexto: Después de las pruebas de aceptación, se realiza una revisión completa del sistema para asegurar que todos los componentes funcionan juntos de manera óptima y que la aplicación es segura y estable.

Ejemplo Real: Se simulan condiciones de uso elevado para asegurarse de que la aplicación puede manejar un gran número de usuarios simultáneos, se prueban las integraciones con sistemas de back-end bancarios y se verifica la seguridad de la aplicación contra posibles ataques cibernéticos.

Las **Pruebas del Sistema** verifican la integración de todos los elementos y se enfocan en la funcionalidad completa del software. Incluyen

- pruebas específicas como pruebas de **recuperación**, para asegurar que el sistema puede continuar operando después de un fallo;
- pruebas de **seguridad**, para verificar que el software está protegido contra amenazas;
- y pruebas de **resistencia**, para evaluar cómo el sistema se comporta bajo condiciones extremas.

Tipo de prueba	Herramienta para Java
UNITARIA	JUnit
INTEGRACIÓN	TestNG
	Hudson
	Continuum
CARGA Y RENDIMIENTO	JMeter
	Jcrawler
ACEPTACIÓN	FitNesse
	Concordion

Documentación para Pruebas

Los estándares de documentación de pruebas en el desarrollo de software son un conjunto de directrices y prácticas recomendadas que tienen como objetivo principal estructurar y estandarizar la forma en que se documentan las pruebas dentro de un proyecto de software. Estos estándares proporcionan un marco que ayuda a los equipos de pruebas a crear documentos consistentes, claros y completos.

https://en.wikipedia.org/wiki/Software_test_documentation

https://in2test.lsi.uniovi.es/gt26/presentations/ISO29119-Presentacion-GT26-20150520.Zaragoza.pdf

Estándar IEEE 829 para un plan de pruebas

Temas IEEE 829	Contenido
Nombre del plan de pruebas: • Ítems a probar • Características a ser probadas • Características que no se van a probar	Esta sección define el alcance del sistema bajo prueba, si los componentes no son completamente nuevos o es una versión incompleta o un incremento. Es importante especificar cuáles características o componentes no van a ser probados.
Enfoque	Provee un overview de la estrategia a utilizar en las pruebas y del proceso de pruebas.
Casos de prueba	Esta sección resume e incorpora por referencia el plan de pruebas, diseño de pruebas individuales, especificaciones de casos de prueba, especificaciones de procedimientos y los resultados de las pruebas.

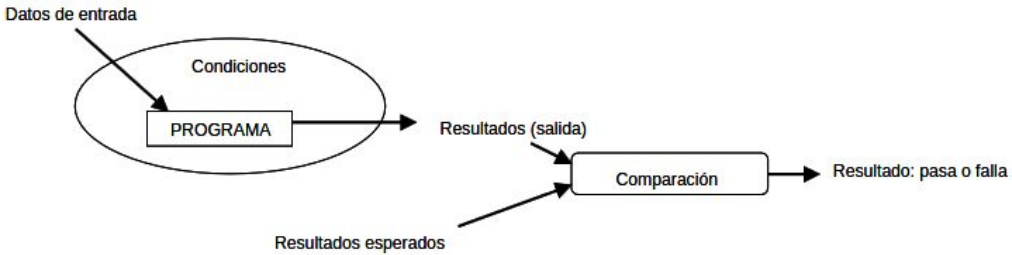
IEEE 829 Subject	Contenido
Tareas	Consiste en la división de las tareas para aplicar las pruebas al sistema
Requerimientos de ambiente	Define las herramientas necesarias para la automatización de las pruebas, hardware, software y lugar para aplicarlas.
Responsabilidades	Define a quién le pertenece cada parte del proceso de pruebas
Personal necesario y entrenamiento	Se necesitará entrenamiento de los inspectores / desarrolladores en cuanto al diseño de pruebas, herramientas, ambiente, etc.
Planificación	Fechas de inicio, hitos, completaciones, etc.
Riesgos y contingencias	¿Qué puede fallar? ¿Qué cosas fueron asumidas? ¿Qué se puede hacer si un problema mayor ocurre?
Aprobación	Determinar quién debe aprobar y revisar el plan

Resumen de los datos esenciales que debería contener la **documentación de pruebas**

- **PLAN DE PRUEBAS**. Describe el alcance, el enfoque, los recursos y el calendario de las actividades de prueba. Identifica los elementos a probar, las características, las tareas, el personal responsable y los riesgos asociados al plan.
- **ESPECIFICACIONES DE PRUEBA**. 3 tipos de documentos: La especificación del diseño de la prueba, la especificación de los casos de prueba y la especificación de los procedimientos de prueba.
- **INFORMES DE PRUEBAS**. Se definen 4 tipos de documentos: un informe que identifica los elementos que están siendo probados, un registro de las pruebas, un informe de incidentes de prueba y un informe resumen de las actividades de prueba

Pruebas de código

1. **Prueba del camino básico**
    - Notación de grafo de flujo
    - Complejidad ciclomática
    - Obtención de los casos de prueba
2. Partición o clases de equivalencia
3. Análisis de valores límite



Las *técnicas de prueba* son mecanismos que los testers utilizan con el objetivo de identificar los casos de prueba con mayor probabilidad de encontrar defectos y obtener la mayor cobertura posible en cuanto a las pruebas de algún sistema

Prueba del camino básico

La prueba estructurada de McCabe, basada en un conjunto básico de caminos opera así:

Paso 1. Convertir el `código` a un ***grafo de control***

Paso 2. Calcular el ***número ciclomático*** y obtener la base de caminos

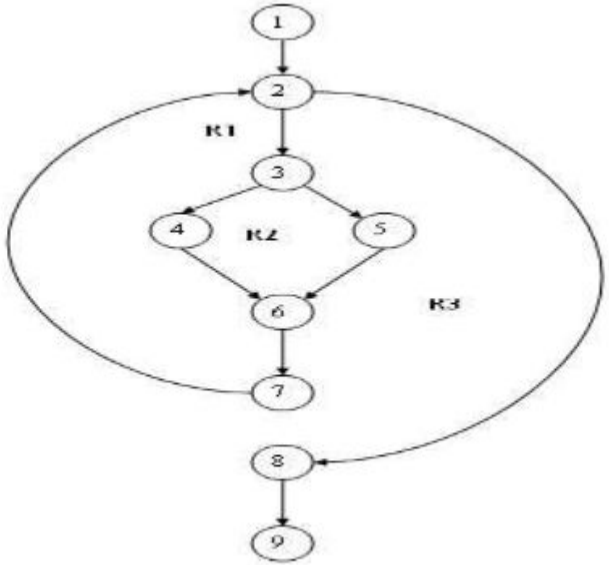
Paso 3. Preparar un ***caso de prueba*** para cada camino básico

La complejidad ciclomática mide la cantidad de rutas de ejecución linealmente independientes a través del código fuente de un programa. En términos más simples, indica el número de caminos diferentes que puede tomar un programa durante su ejecución.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main( void )
{
        int valor=0;
        int intentos=0;
        int aleatorio=0;
        /* Inicializar y asignar un numero aleatorio.*/
        srand(time(NULL));
        aleatorio = rand() % 101;
        /* Encontrar el numero*/
(1)     printf("Acierte el numero que he pensado (0-100): ");
        scanf("%i", &valor);
        intentos = intentos + 1;
        /* Intentarlo mientras no acierte*/
        while( valor != aleatorio )  (2)
        {
                /* Comprobar si es mayor o menor.*/
                if( valor < aleatorio) (3)
                {
                        printf ("Mi numero es mayor\n");  (4)
                }
                else
                {
                        printf ("Mi numero es menor\n"); (5)
                }  (6)
                /* Pedir un nuevo intento*/
                printf("Intentalo de nuevo: ");
                scanf("%i", &valor);                          7
                intentos = intentos + 1;

        }  (8)

9       /*Mostrar mensaje de acierto*/
        printf("CORRECTO !!! Has acertado tras %i intentos\n", intentos);
        system("PAUSE");
}
```

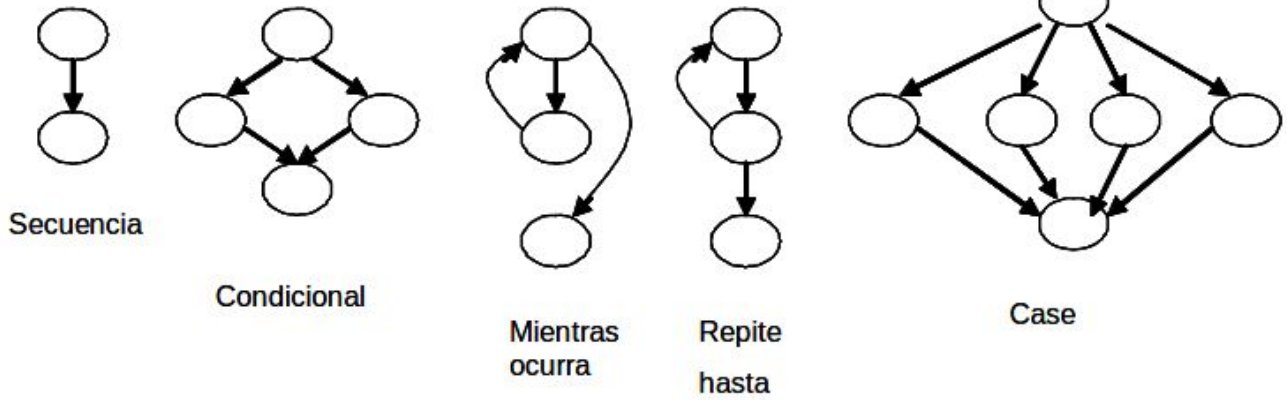


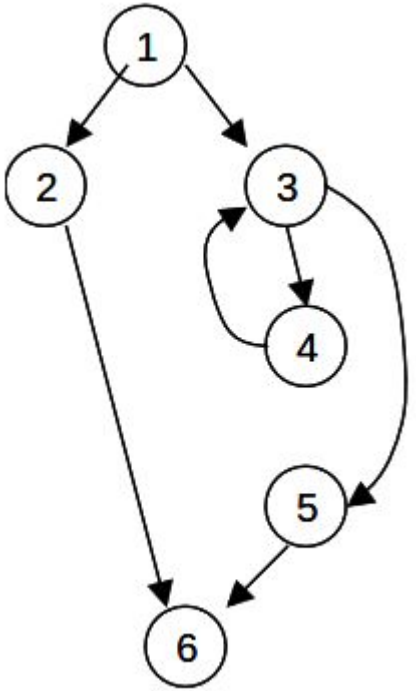
Cálculo de V (G):
V (G) = 3 regiones = 3 → V (G) = 10A – 9N + 2 → V (G) = 2 NP + 1 = 3
Conjunto de Caminos Básicos: Habrá tantos caminos básicos como grados de complejidad posee el código.
En este caso tenemos tres caminos básicos:

- o C1: 1-2-3-4-6-7-2-8-9
- o C2: 1-2-3-5-6-7-8-9
- o C3: 1-2-8-9

Prueba del camino básico

Cada ***nodo*** representa una o más sentencias, sin bifurcaciones, en pseudocódigo o código fuente





Prueba del camino básico

El modelo estructural construye un grafo equivalente al artefacto bajo estudio, ya sea un procedimiento o un método, donde se cumple:

a) existe un solo ***nodo*** inicial

b) existe un solo ***nodo*** final

c) cada ***nodo*** representa una secuencia de instrucciones, que puede ser vacía.

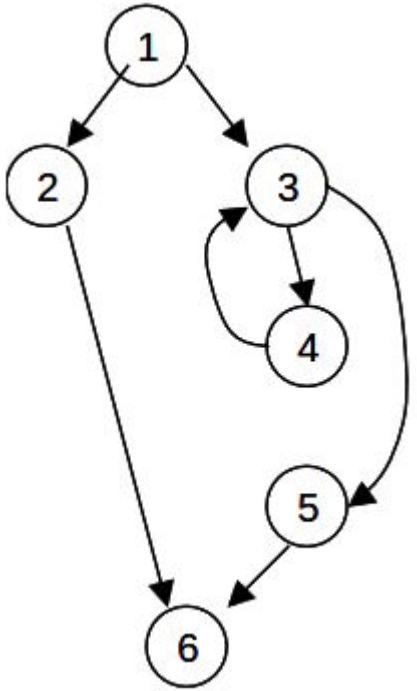
d) La relación entre los ***nodos*** es una relación de “el control pasa a”.

e) Un ***nodo*** puede tener uno o dos sucesores,

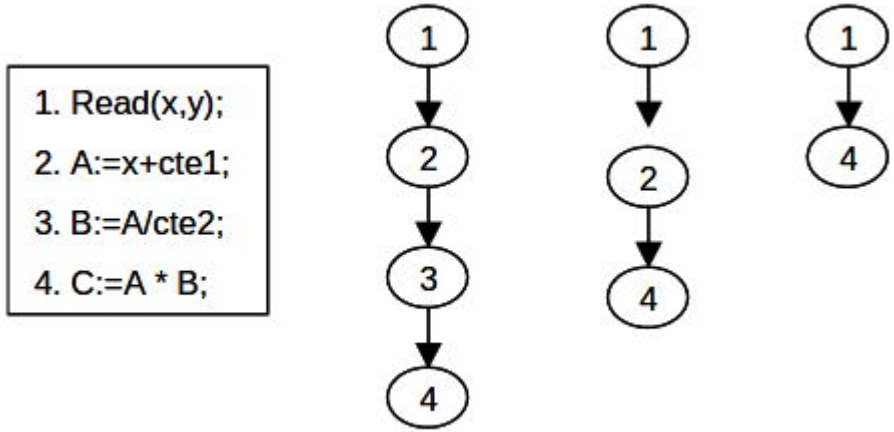
f) Un ***nodo*** puede tener uno o muchos antecesores

g) Un ***nodo*** tendrá dos sucesores si existen dos caminos posibles, dependiendo de una condición

h) Los casos de for, while, until y case se pueden reducir a grupos de ***nodos*** con dos sucesores

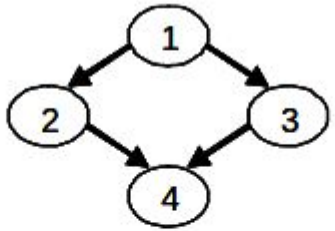


Representación de una secuencia de instrucciones

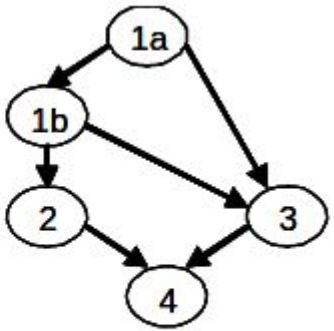


Representación de una bifurcación de instrucciones

```
1. if (a<3.47 and zx =0)
2. then sd:=2
3. else sd:=1;
4. print (sd);
```



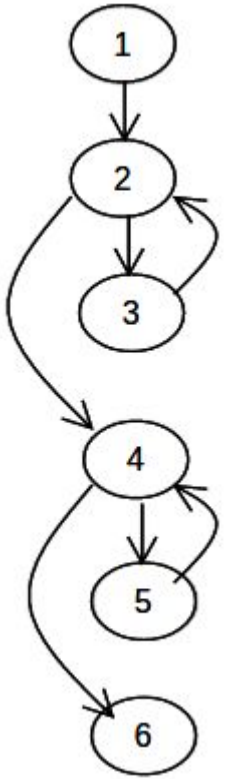
Conjunción
"en bruto"



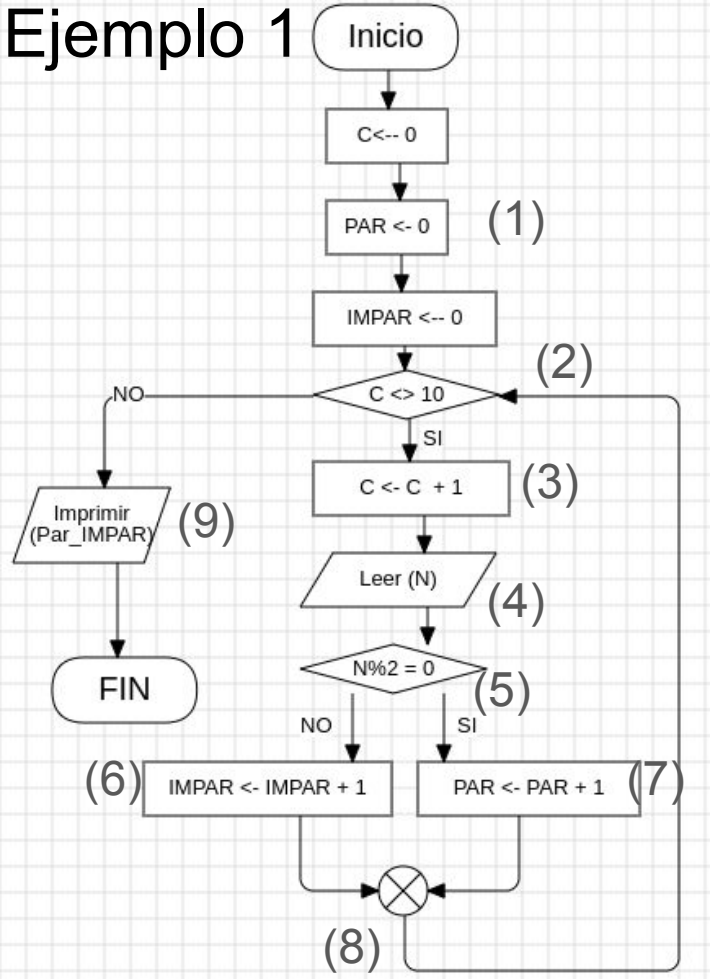
Conjunción
expandida

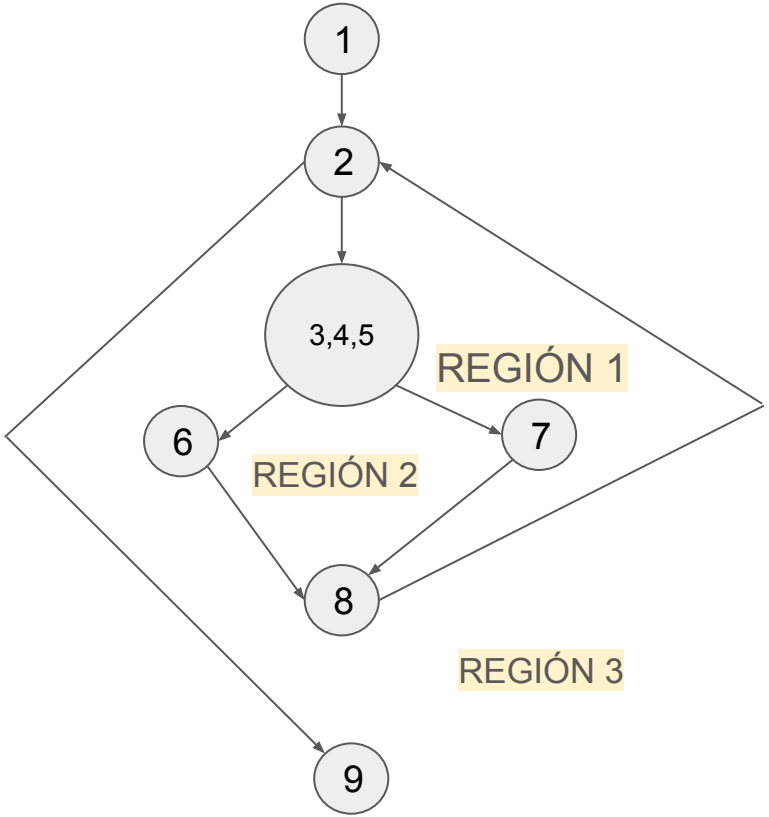
Representación de una bucle de instrucciones

```
1  i=1     j=1
2  While not eof(a)
3      lee ARRA[i]     i=i+1
4  While not eof(b)
5      lee ARRB[j]     j=j+1
6  regresa
```



Ejemplo 1





Un programa que lee 10 números de teclado y muestra cuántos números leídos son pares y cuántos son impares .

Ejemplo 1

Cada círculo del grafo se llama **nodo**. Representa una o más sentencias procedimentales. Un solo nodo se puede corresponder con una secuencia de símbolos del proceso y un rombo de decisión. Un ejemplo es el nodo numerado como 3,4,5.

Las flechas del grafo de flujo se denominan **aristas** o enlaces y representan el flujo de control, como en el diagrama de flujo. Una arista termina en un nodo, aunque el nodo no tenga ninguna sentencia procedimental; es el caso del nodo numerado como 8.

Las áreas delimitadas por aristas y nodos se llaman **regiones**, el área exterior del grafo es otra región más. En el ejemplo se muestran 3 regiones, 8 aristas y 7 nodos.

El nodo que contiene una condición se llama nodo predicado y se caracteriza porque de él salen dos o más aristas. En el ejemplo se muestran dos nodos predicado, el representado por el número 2 y el representado por 3,4,5.

Complejidad ciclomática

La complejidad ciclomática es una métrica del software que proporciona una medida cuantitativa de la complejidad lógica de un programa. En el contexto del método de prueba del camino básico, la complejidad ciclomática establece el número de caminos independientes del conjunto básico de caminos de ejecución de un programa y por lo tanto el número de casos de prueba que se deben ejecutar para asegurar que cada sentencia se ejecuta al menos una vez.

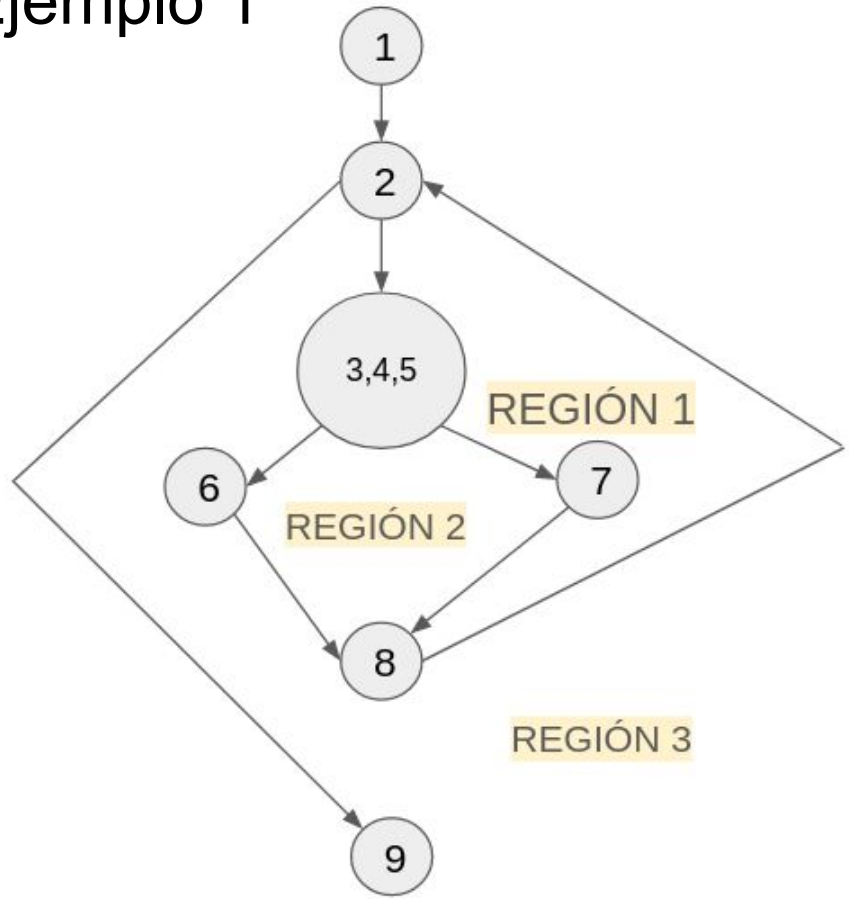
La **complejidad ciclomática** V(G) puede calcularse de tres formas:

1. V(G) = Número de regiones del grafo
2. V(G) = Aristas - Nodos + 2
3. V(G) = Nodos predicado + 1

Complejidad ciclomática V(G)	**Evaluación del riesgo**
Entre 1 y 10	Programas o métodos sencillos, sin mucho riesgo
Entre 11 y 20	Programas o métodos más complejos, riesgo moderado
Entre 21 y 50	Programas o métodos complejos, alto riesgo
Mayor que 50	Programas o métodos no testeables, muy alto riesgo

El valor V(G) nos va a dar el número de caminos independientes del conjunto básico de un programa. Un camino independiente será un camino en el cual se introducirán nuevas sentencias de proceso o una condición. Referente a los diagramas de flujo, estarán constituidos por al menos una arista que no ha sido recorrida anteriormente a la definición del camino.

Ejemplo 1



La complejidad ciclomática es 3

1. V(G) = Número de regiones del grafo = 3
2. V(G) = Aristas - Nodos + 2 = 8 - 7 + 2 = 3
3. V(G) = Nodos predicados + 1 = 2 + 1 = 3

Un conjunto de caminos independientes será:
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

Camino	Caso de prueba	Resultado esperado
1	Escoger algún valor de C tal que NO se cumpla la condición C<>10 C=10	Visualizar el número de pares y el de impares
2	Escoger algun valor de C tal que Sí se cumjpla la condición C<>10. Escoger algún valor de N tal que NO se cumpla la condición N%2=0 C=1, N=5	Contar número de impares
3	Escoger algún valor de C tal que sí se cumpla la condición C<>10 Escoger algún valor de N t<al que sí se cumpla condición N%2 = 0 C=2, N = 4	Contar números de pares

Ejemplo 2

```
static void visualizarMedia(float x, float y)
{
    float resultado = 0;

     if((x < 0) || (y <0)) {
          //Imprime error
          System.out.println("X e Y deben ser positivos");
    }else {
         //Calcula e Imprime
         resultado = (x+y)/2;
         System.out.println("La media es: " + resultado);
    }
}
```

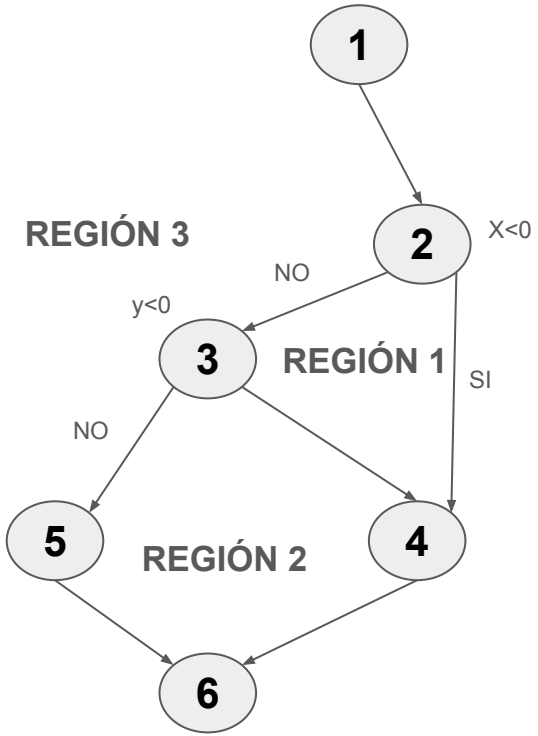
Ejemplo 2

```
static void visualizarMedia(float x, float y)
{
    float resultado = 0;                                    (1)

     if((x < 0) || (y <0)) {                                (2) (3)
         //Imprime error
         System.out.println("X e Y deben ser positivos");   (4)
    }else {
         //Calcula e Imprime
         resultado = (x+y)/2;
         System.out.println("La media es: " + resultado);   (5)
    }
} (6)
```

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

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



Ejemplo 2

Camino	Caso de pruebas	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 visualizaMedia(4,5)	Visualiza: “La media es: 4,5”
camino 2: 1-2-4-6	Escoger alún X tal que Sí se cumpla la condición X<0 (Y puede ser cualquier valor) x= - 4, y=5 visualizaMedia(-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 visualiarMedia(4, - 5)	Visualiza: “X e Y deben ser positivos”

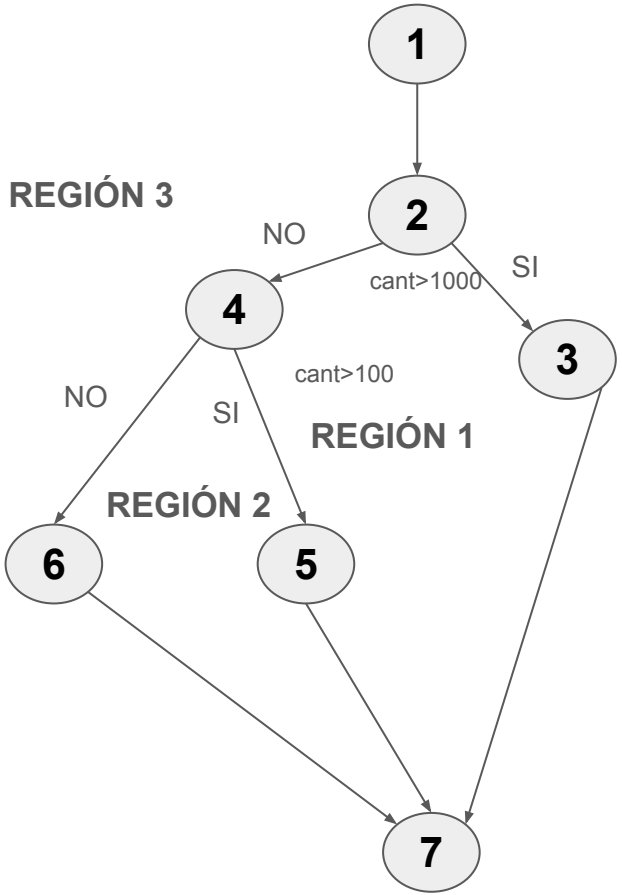
Ejemplo 3

```
1  Algoritmo ENDES
2      Leer cant
3      Leer pvp
4      importe ← cant * pvp
5      Si cant > 1000 Entonces
6          dto ← importe * 0.1
7      Sino
8          Si cant > 100 Entonces
9              dto = importe * 0.05
10         Sino
11             dto = 0
12         FinSi
13     FinSi
14     importeTotal ← importe - dto
15     Imprimir "Total : " , importeTotal
16 FinAlgoritmo
```

Ejemplo 3

```
1  Algoritmo ENDES
2      Leer cant
3      Leer pvp
4      importe ← cant * pvp
5      Si cant > 1000 Entonces
6          dto ← importe * 0.1
7      Sino
8          Si cant > 100 Entonces
9              dto = importe * 0.05
10         Sino
11             dto = 0
12         FinSi
13     FinSi
14     importeTotal ← importe - dto
15     Imprimir "Total : " , importeTotal
16 FinAlgoritmo
```

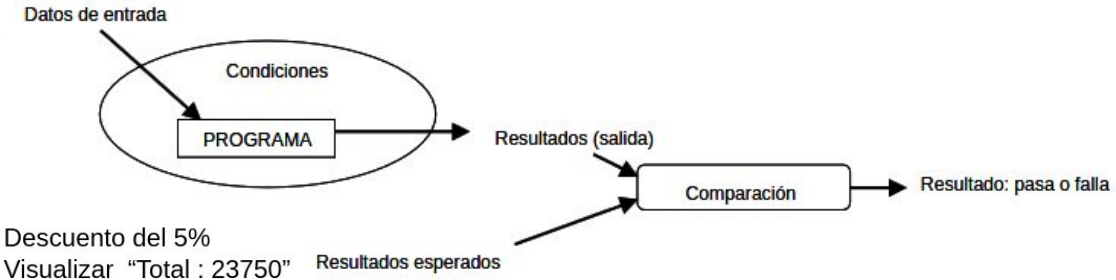
(1) (2) (3) (4) (5) (6) (7)



Ejemplo 3

Camino	Caso de pruebas	Resultado esperado
camino 1: 1-2-3-5-6	cant=200, pvp=125	Descuento del 5% Visualizar “Total : 23750”
"camino 2: 1-2-4-6	cant 5, pvp=125	Descuento 0. Visualizar “Total 625”
camino 3: 1-2-3-4-6	cant = 2000, pvp = 125	Descuento 10% Visualizar: “Total 225000

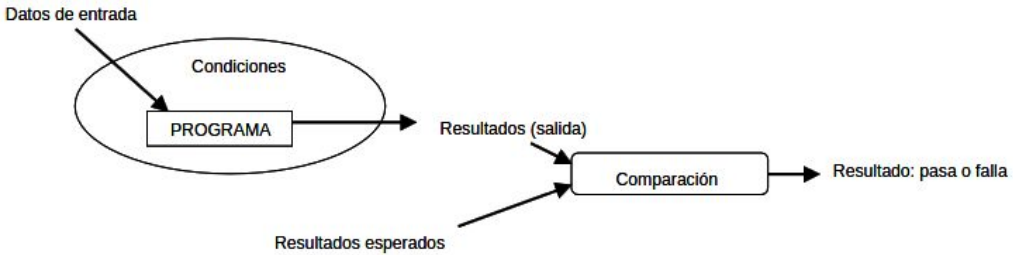
cant=200, pvp=125



Pruebas de código

1. Prueba del camino básico
   - Notación de grafo de flujo
   - Complejidad ciclomática
   - Obtención de los casos de prueba
2. **Partición o clases de equivalencia**
3. Análisis de valores límite

*Prueba de **Caja Negra***



Las *técnicas de prueba* son mecanismos que los testers utilizan con el objetivo de identificar los casos de prueba con mayor probabilidad de encontrar defectos y obtener la mayor cobertura posible en cuanto a las pruebas de algún sistema

Clases de equivalencia

Una clase de equivalencia es un conjunto de valores de entrada para los cuales se espera que el sistema se comporte de la misma manera.

Estas clases se utilizan para simplificar las pruebas, reduciendo el número total de casos de prueba necesarios, al considerar que una prueba para un valor representativo de una clase será suficiente para todo el conjunto.

En el contexto de la partición de equivalencia, las clases se dividen típicamente en dos categorías:

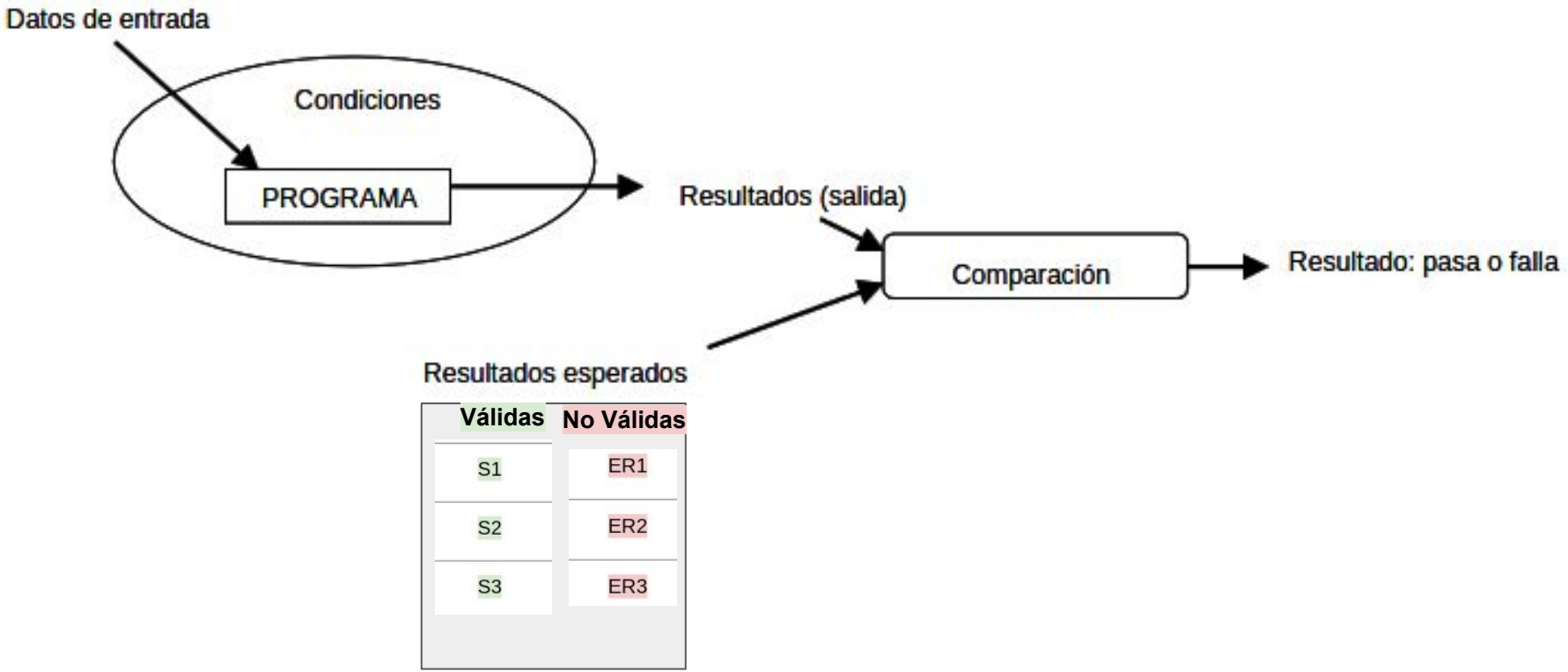
1. **Clases de Equivalencia Válidas**: Contienen valores de entrada que se espera sean aceptados por el sistema. Por ejemplo, si un campo de formulario acepta edades entre 18 y 60 años, entonces cualquier edad dentro de este rango pertenecería a una clase de equivalencia válida.
2. **Clases de Equivalencia No Válidas**: Incluyen valores de entrada que se espera sean rechazados o que causen errores en el sistema. Siguiendo el mismo ejemplo, las edades menores de 18 años o mayores de 60 años formarían clases de equivalencia no válidas.

Resultados esperados:

Válidas

No Válidas

Condiciones de entrada		
Empleado	Departamento	Oficio



2. Partición o clases de equivalencia

La técnica de partición equivalente es una metodología utilizada en las **pruebas de caja negra** de software, que se enfoca en dividir el dominio de entrada de un programa en clases de equivalencia.

Cada **clase** representa *un conjunto de* estados válidos o no válidos para las condiciones de entrada del programa.

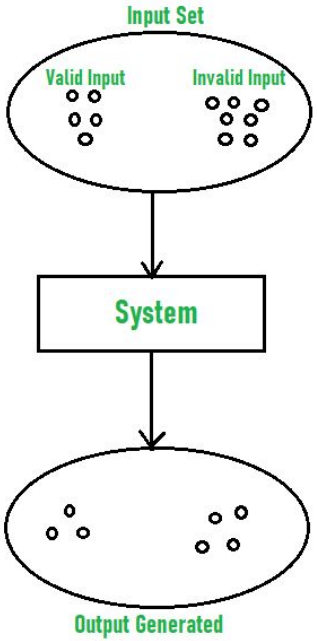
La idea principal es que, en lugar de probar cada valor de entrada individualmente, se pueden probar representantes de cada **clase de equivalencia** para obtener resultados efectivos.

Existen ciertas directrices para definir estas clases de equivalencia en la partición equivalente:

1. Si la condición de entrada es un rango, se definen una clase de equivalencia válida y dos no válidas.
   - Ejemplo de rango: la nota debe tener un valor entre 1 y 10
2. Si la entrada es un valor específico, también se definen una clase válida y dos no válidas.
   - Ejemplo de valor específico: el número departamento puede ser blanco o tener 2 dígitos
3. Si la entrada es un miembro de un conjunto, se define una clase válida y una no válida.
   - Ejemplo: El curso puede tener los siguientes valores: “1DAW”, “2DAM”, “1SMR”, “2SMR”
4. Para una condición de entrada lógica (como un valor booleano), se establece una clase válida y una no válida.
   - Ejemplo: el salario debe ser > que 0

2. Partición o clases de equivalencia

Condiciones de entrada	Nº de clases de equivalencia válidas	Nº de clases de equivalencia no válida
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
Valor específico	1 CLASE VÁLIDA ● Contempla dicho valor	2 CLASES NO VÁLIDAS ● Un valor por encima ● Un valor por debajo
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
Lógica	1 CLASE VÁLIDA ● Una clase que cumpla la condición	1 CLASE NO VÁLIDA ● Una clase que no cumpla la condición



La tabla resume el número de clases de equivalencia válidas y no válidas que hay que definir para cada tipo de condición de entrada

Ejemplo clases de equivalencia

Se va a realizar una entrada de datos de un empleado por un formulario en pantalla gráfica, se definen 3 campos de entrada

La aplicación acepta los datos de esta manera:

- *Empleado*: número de 3 dígitos que no empiece por 0
- *Departamento*: en blanco o número de dos dígitos
- *Oficio*: Analista, Diseñador, Programador o ‘Elige oficio’

Ejemplo App

<SALIDAS ESPERADAS>

Ejemplo clases de equivalencia

Si la entrada es correcta el programa asigna un salario (que se muestra en pantalla) a cada empleado según estas normas:

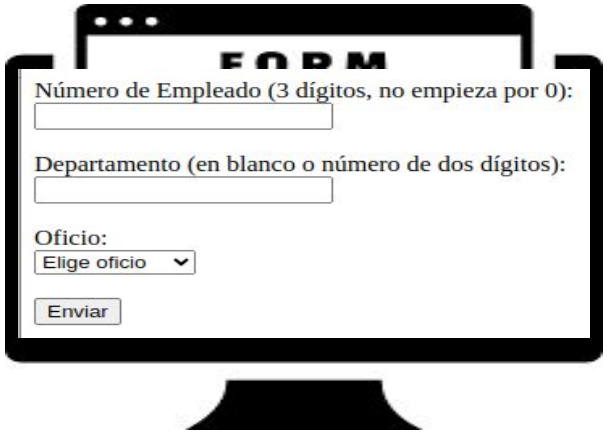
- **S1** si el ***Oficio*** es '*Analista*' se asigna 2500
- **S2** si el ***Oficio*** es '*Diseñador*' se asigna 1500
- **S3** si el ***Oficio*** es '*Programador*' se asigna 2000

Si la entrada no es correcta el programa muestra un mensaje indicando la entrada incorrecta:

- **ER1** si el Empleado no es correcto
- **ER2** si el Departamento no es correcto
- ***ER3*** *si no se ha elegido el Oficio*

ENTRADAS FORMULARIO:

- ***Empleado***:
  número de 3 dígitos que no empiece por 0
- ***Departamento***:
  en blanco o número de dos dígitos
- ***Oficio***:
  Analista, Diseñador, Programador o 'Elige oficio'



Ejemplo App

Clases de equivalencia

Para representar las clases de equivalencia para cada condición de entrada se puede usar una tabla

Condición de entrada	Clases de equivalencia	Clases Válidas	COD	Clases no Válidas	COD
Empleado	Rango	100>= empleado <= 999	V1	Empleado <= 099 Empleado > 999	NV1 NV2
Departamento	Lógica	En blanco	V2	No es un número	NV3
	Valor	Cualquier número de dos dígitos	V3	Número de más de 2 dígitos Número de menos de 2 dígitos	NV4 NV5
	Rango	10 >= departamento <= 99	V3*	Dep > 99 Dep < 10	NV4 * NV5 *
Oficio	Miembro de un conjunto	Oficio = 'Programador'	V4	Oficio = "Elige un oficio"	NV6
		Oficio = 'Analista'	V5		
		Oficio = 'Diseñador'	V6		

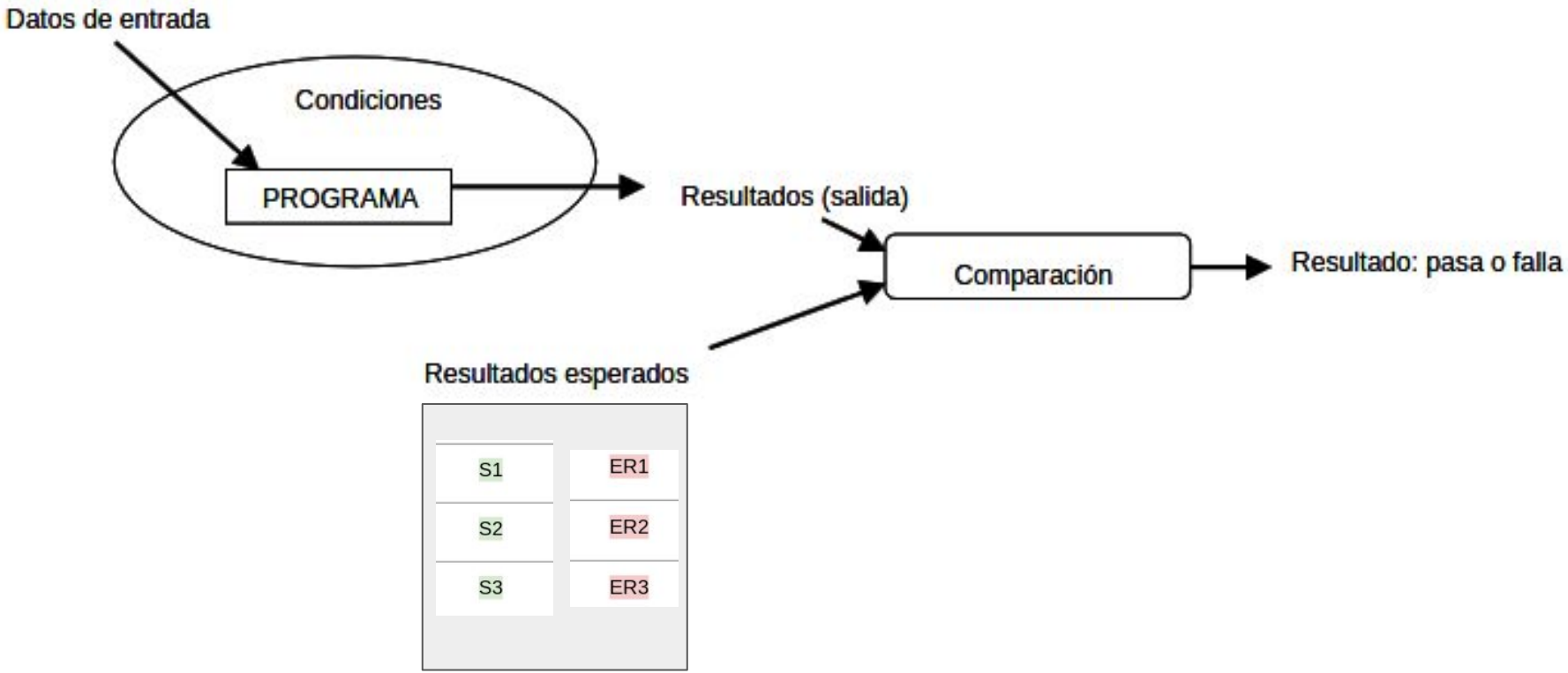
ENTRADAS FORMULARIO:
- ***Empleado***: número de 3 dígitos que no empiece por 0
- ***Departamento***: en blanco o número de dos dígitos
- ***Oficio***: Analista, Diseñador, Programador o 'Elige oficio'

A partir de esta tabla se genera los casos de prueba

Casos de prueba por cada *resultado esperado*

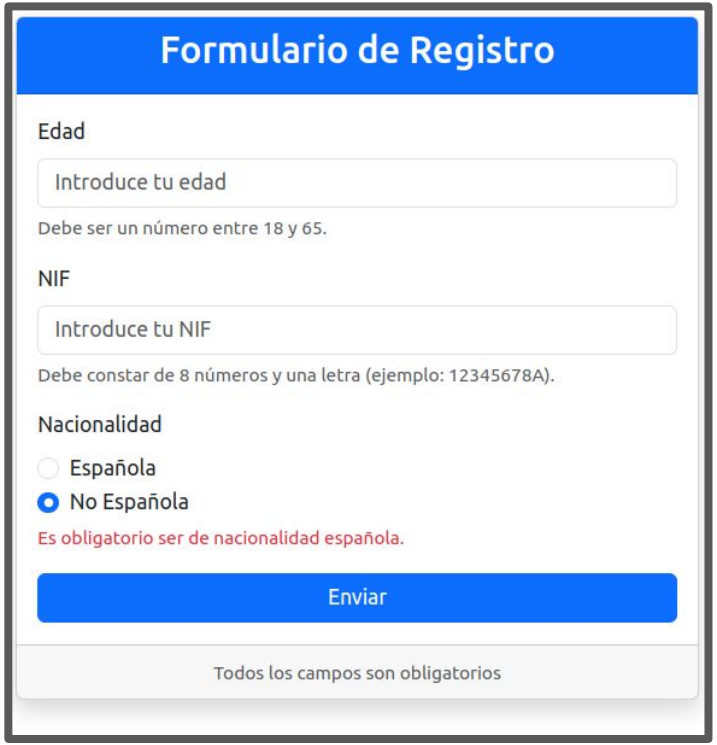
CASO DE PRUEBA	Clase de equivalencia [COD]	Condiciones de entrada			Resultados esperados
		Empleado	Departamento	Oficio	
CP1	V1, V2, V5	250		Analista	S1
CP2	V1, V3, V5	450	30	Diseñador	S2
CP3	V1, V3, V4	200	20	Programador	S3
CP4	V1,V2,V4	220		Programador	S3
CP5	NV1, V2, V6	90	35	Diseñador	ER1
CP6	V1, NV3, V5	100	AD	Analista	ER2
CP7	V1, V2, NV8	300		‘Elige oficio’	ER3
CP8	V1, NV4, V6	345	123	Diseñador	ER2
…	…	…	…	…	…

Condiciones de entrada		
Empleado	Departamento	Oficio



Prueba de particiones o clases de equivalencia

En una aplicación, la persona usuaria debe introducir su edad, que debe ser un número entre 18 y 65; su NIF, que debe constar de 8 números y una letra; y, además, debe indicar si tiene o no nacionalidad española, de forma que tener nacionalidad española es un requisito.



Prueba de particiones o clases de equivalencia

Condición de entrada	**Clases Válidas**	**Clases no válidas**
edad	• 18 ≤ edad ≤ 65 (V1)	• edad < 18 (NV1) • edad > 65 (NV2) • No es un número (NV3)
NIF	• Una cadena de 9 caracteres compuesta por 8 números y una letra (V2)	• < 9 caracteres (NV4) • > 9 caracteres (NV5) • El último carácter no es una letra (NV6) • Alguno de los 8 primeros caracteres no es un número (NV7)
nacionalidad	• Española (V3)	• No española (NV8)

Casos de pruebas

Con clases de equivalencia válidas

edad	NIF	Nacionalidad	Clases incluidas
35	32323267G	Española	V1, V2, V3

Condición de entrada	Clases Válidas	Clases no válidas
edad	• 18 ≤ edad ≤ 65 (V1)	• edad < 18 (NV1) • edad > 65 (NV2) • No es un número (NV3)
NIF	• Una cadena de 9 caracteres compuesta por 8 números y una letra (V2)	• < 9 caracteres (NV4) • > 9 caracteres (NV5) • El último carácter no es una letra (NV6) • Alguno de los 8 primeros caracteres no es un número (NV7)
nacionalidad	• Española (V3)	• No española (NV8)

Condición de entrada	Clases Válidas	Clases no válidas
edad	• 18 ≤ edad ≤ 65 (V1)	• edad < 18 (NV1) • edad > 65 (NV2) • No es un número (NV3)
NIF	• Una cadena de 9 caracteres compuesta por 8 números y una letra (V2)	• < 9 caracteres (NV4) • > 9 caracteres (NV5) • El último carácter no es una letra (NV6) • Alguno de los 8 primeros caracteres no es un número (NV7)
nacionalidad	• Española (V3)	• No española (NV8)

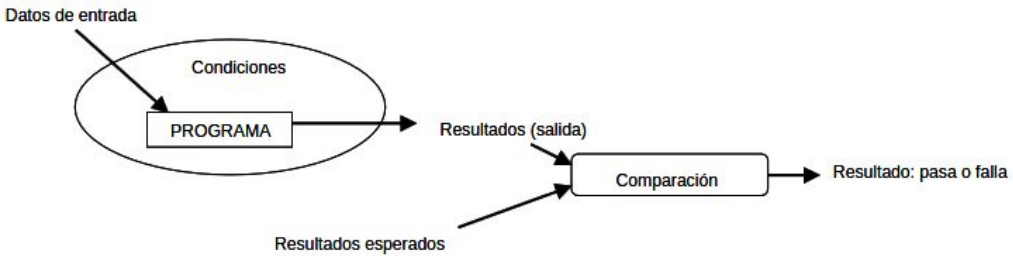
Casos de pruebas

Con clases de equivalencia no válidas

edad	NIF	Nacionalidad	Clases incluidas
16	12345678A	Española	NV1, V2, V3
73	12345678A	Española	NV2, V2, V3
1A	87654321B	Española	NV3, V2, V3
18	5678A	Española	V1, NV4, V3
18	9123456789B	Española	V1, NV5, V3
18	123456788	Española	V1, NV6, V3
18	A2345678B	Española	V1, NV7, V3
18	12345678A	No Española	V1, V2, NV8

Pruebas de código

1. Prueba del camino básico
   - Notación de grafo de flujo
   - Complejidad ciclomática
   - Obtención de los casos de prueba
2. Partición o clases de equivalencia
3. **Análisis de valores límite**



Las *técnicas de prueba* son mecanismos que los testers utilizan con el objetivo de identificar los casos de prueba con mayor probabilidad de encontrar defectos y obtener la mayor cobertura posible en cuanto a las pruebas de algún sistema

Análisis de valores límite

Este análisis se basa en la hipótesis de que suelen ocurrir más errores en los valores extremos de los campos de entrada. Complementa a la técnica anterior. Además, no solo estará centrado en las condiciones de entrada, sino que definen también las clases de salida.

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, y 5
Antigüedad	De 0 a 25 (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

Video


Técnica de partición equivalente. Ejemplo
UNIVERSITAT POLITÈCNICA DE VALÈNCIA
Diseño de casos de prueba para cubrir una y solo una clase de equivalencia inválida cada vez (2, 3, 4, 6, 7 y 12)
Código de área
Clave identificativa
Orden
Clase/s inválida/s cubierta/s
150
1100
0
Teléfono
Viajes
1, 5, 12

@TODO

- JUNIT, (Selenium)?
- Herramientas de depuración

Anexo :: DOCUMENTACIÓN de un PROGRAMA

A) Guía técnica

B) Guía de uso

C) Guía de instalación

Anexo :: DOCUMENTACIÓN de un PROGRAMA

A) Guía técnica

a. Información que contiene:

El diseño de la aplicación.

La codificación de los programas.

Las pruebas realizadas.

b. Dirigido a:

Personal técnico en informática (analistas y programadores)

c. Objetivo:

Facilitar un correcto desarrollo, realizar correcciones en los programas y permitir un mantenimiento futuro.

Anexo :: DOCUMENTACIÓN de un PROGRAMA

B) Guía de uso

a. Información que contiene:

Descripción de la funcionalidad de la aplicación.

Forma de comenzar a ejecutar la aplicación.

Ejemplos de uso del programa.

Requerimientos software de la aplicación.

Soluciones de los posibles problemas que se pueden presentar.

b. Dirigido a:

Usuarios que van a utilizar la aplicación (clientes).

c. Objetivo:

Dar a los usuarios finales toda la información necesaria para utilizar la aplicación.

Anexo :: DOCUMENTACIÓN de un PROGRAMA

C) Guía de instalación

a. Información que contiene:

Toda la información necesaria sobre puesta en marcha, explotación y seguridad del sistema.

b. Dirigido a:

Personal informático responsable de la instalación, en colaboración con los usuarios que van a usar la aplicación (clientes).

c. Objetivo:

Dar toda la información necesaria para garantizar que la implantación de la aplicación se realice de forma segura, confiable y precisa