APUNTS ENTORNOS DE DESARROLLO 3r TRIMESTRE

Unidad 4 - Pruebas de unidad

1.- Técnicas de diseño

Pruebas de caja blanca {

Validar la estructura **interna** del programa.

Hay que conocer los **detalles** del código.

Prueba ideal → Probar todos los posibles **caminos** de ejecución.

Solo las puede realizar el **programador**.

}

Pruebas de caja negra {

Validar el **funcionamiento**.

Hay que conocer la **funcionalidad** esperada.

Prueba ideal → Probar todas las posibles **entradas y salidas**.

Las puede realizar el **programador y cliente**.

}

2.- Estrategias de pruebas del software

En el contexto de una espiral {

En orden:

Prueba de unidad {

Se centra en los módulos del **código fuente**.

Se utilizan pruebas de **caja blanca y negra**.

Herramientas → **JUnit**.

}

Prueba de integración {

Se juntan los módulos en una **estructura** de acuerdo con lo que dicta el **diseño**.

Se trata de comprobar si funcionan **juntos**.

}

Prueba de validación {

En el **entorno real** con el usuario final (alfa, beta...).

Se validan los **requisitos** establecidos en el análisis.

Se utilizan pruebas de **caja negra**.

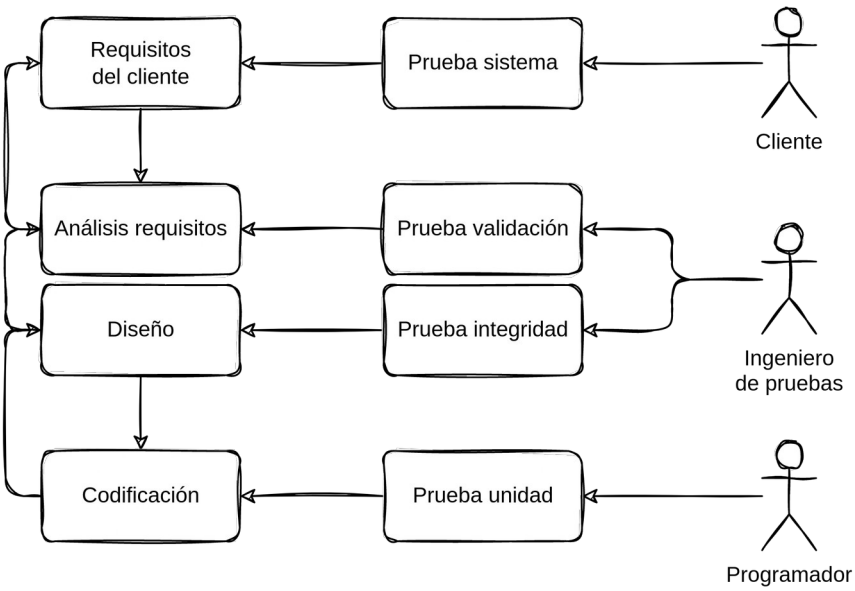
}

Prueba de sistema {

Se prueba **todo** el programa, buscando **forzar** fallos y vulnerabilidades.

}

}



3.- Complejidad ciclomática

Complejidad ciclomática V(G) {

Cuántos **caminos diferentes** podemos tener.

Fórmula {

V(G) = P + 1

P son los nodos que representan **condiciones.**

}

}

4.- Prueba equivalencia

Agrupamos las **posibles entradas** en clases de equivalencia.

Deben ser creadas para encontrar entradas **válidas y no válidas.**

Ejemplo:

Condiciones	Válidas	No válidas
Descripción (alfanumético 8 caracteres)	1 clase 8 caracteres	3 clases menos de 8 caracteres más de 8 caracteres no alfanumérico

5.- Valores frontera

Se trata de probar los valores **límite** de las clases **equivalentes.**

En un **rango** de valores {

Se proabrá el **mínimo**, **máximo** y los valores justo por **encima** o **debajo.**

[1 - 10] → Se probará: 1, 10, 0 y 11.

}

Especificaciones	Casos a probar
Descripción (entre 1.0 y 15.0)	0.9, 1.0, 15.0, 15.1

Unidad 5 - JUnit

1.- Que es JUnit

JUnit es un **framework** que poermite realizar **pruebas** de unidad en clases Java.

Para esto, se crea una clase con métodos de prueba.

Mediante **asserts** se verificará el funcionamiento del programa {

Los métodos assertXXX() permiten **comprobar** si la salida de un método es la esperada.

Són de tipo **void**.

assertNull(), assertNotNull(), assertTrue(), assertFalse(), assertEquals().

}

2.- Inicializadores y Finalizadores:

Método **setUp()** {

Se ejecuta **antes** de cada caso de prueba.

}

Método **tearDown()** {

Se ejecuta **después** de cada caso de prueba.

}

Unidad 6 - Sistemas de control de versiones

1.- Que es un CVS

Un sistema control de **versiones** es una **herramienta** utilizada en la fase de codificación, integración y mantenimiento que permite {

Guardar un historico de la **evolución** del código.

Solventar **conflictos** cuando hay más de una persona con el mismo fichero.

}

2.- Que es Git

Git es el CVS libre más **utilizado {**

Repositorio {

Lugar donde se **almacenan** todos los datos del CVS.

Git crea un conjunto de **instantáneas** cada vez que hay algún cambio, los ficheros no modificados no se almacenan de nuevo.

Remoto {

En el **servidor**.

Compartido por todo el equipo.

}

Local {

En el PC.

Individual para cada usuario.

}

}

}

3.- Áreas de trabajo

Cada una de las **secciones** del proyecto {

Directorio de trabajo (Working Directory) {

Ficheros sobre los que **trabajamos.**

}

Área de preparación (Staging Area) {

Ficheros **modificados** incluidos en el siguiente **commit.**

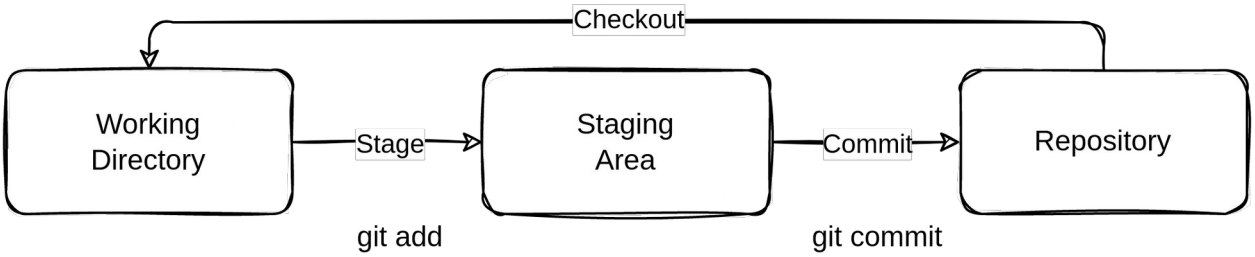
}

Directorio de Git (Repository) {

Lugar donde se **almacenan** todos los datos del CVS.

}

}



4.- Comandos Git

Commit {

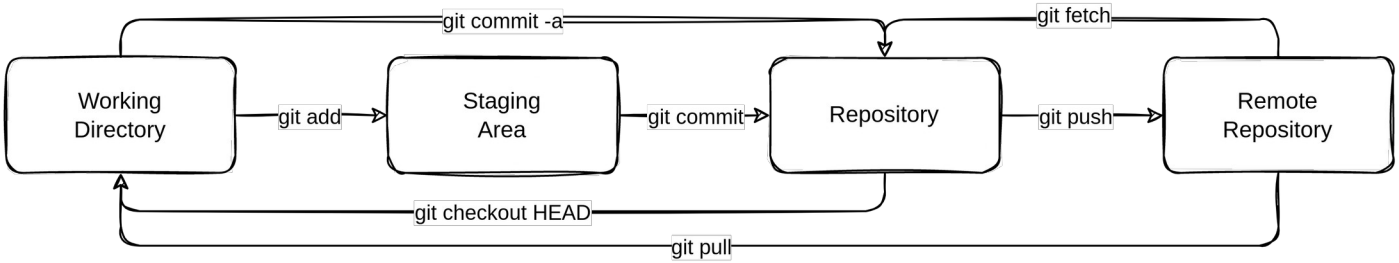
Recoge los **cambios** de los archivos del área de **preparación** y los guarda en una **instantánea** del repositorio.

}

HEAD {

Es una **referencia** al **commit** en el que estamos trabajando.

}



5.- Ramas

Permite **aislar** distintas evoluciones **paralelas** del desarrollo.

La rama principal se llama **MASTER.**

Comandos {

git branch <nombre rama> → **Crear** rama

git branch → **Ver** ramas.

git checkout <nombre rama> → **Cambiar** de rama

git merge <nombre rama> → **Fusionar** rama

git branch -d <nombre rama> → **Borrar** rama.

}

Unidad 7 - JavaDoc

1.- Que es JavaDoc

JavaDoc permite generar documentación de API en formato HTML a partir de comentarios en el código fuente de Java.

JavaDoc utiliza etiquetas HTML y palabras reservadas precedidas por "@".

```java
/**
 * Suma un plus al salario del empleado si el empleado tiene más de 40 años
 *
 * @param sueldoPlus Aumento del sueldo
 * @return <ul>
 *                <li>true: se suma el plus al sueldo</li>
 *                <li> false: no se suma el plus al sueldo</li>
 *          </ul>
 */
public boolean plus(double sueldoPlus) {
      boolean aumento = false;
      if (edad > 40 && compruebaNombre()) {
            salario += sueldoPlus;
            aumento = true;
      }
      return aumento;
}
```