TIPOS DE ERRORES.

Mistake -> Error humano.
Bug, defect -> Error de código.
Failure -> Error de ejecución.

Actividades para prevenir errores (PROCESO DE PRUEBAS)(en orden)

1) PLANIFICACION:
Marcarnos los objetivos de las pruebas. Saber que pruebas quiero y no hacer.

2) DISEÑO de las pruebas:
El mas importante, nuestras pruebas van a ser mejores o peores dependiendo de este proceso. Escoger de forma adecuada entradas de las cuales obtenemos salidas esperadas.

3) IMPLEMENTACION y ejecución de pruebas:
Haremos otro programa (nuevo código), añadiendo código para comparar el esperado con el real y nos diga si si o si no. Hay que utilizar cada cosa cuando toca, no todo vale para todo (lógico)

4) EVALUACION del proceso de pruebas e informe:
Comprobar que de verdad hemos conseguido lo que deberíamos conseguir.

Por que probamos?
Porque somos humanos y cometemos errores. Nos consumirán un tiempo nada despreciable.

EXITO (mejor o peor tester)
Entregar a tiempo.
Que le cueste al cliente lo que le iba a costar
Que haga lo que el cliente te ha dicho que tiene que hacer.

PRINCIPIOS (Verdades universales) de las pruebas.

Trabajo tester consiste en demostrar la presencia de errores (demostrar que hay algo que esta mal).

No podemos probar con todas las posibles entradas (ej. si una entrada es un entero, habría que probar con todos los enteros).

Hay que probar cuanto antes, antes detectaremos el problema. Cuanto más se tarde, mas tardaremos en reparar.

Clustering de defectos, los defectos suelen estar agrupados porque cada desarrollador suele implementar una parte, entonces todo lo que le sigue a un fallo suele conllevar otros. Si hemos encontrado errores hay que escarbar mas.

La paradoja del testicida, los test deben revisarse, mi código de test no ha evolucionado al mismo nivel que el código.

El código puede no tener errores pero ¿satisface las exigencias del cliente? No es solo verificar el código. Si el cliente quiere ‘A’ hay que darle ‘A’.

COMO HACER LAS PRUEBAS. Dos procesos

Testing(pruebas) y **Debugging**(depuracioón)

A los tester no nos preocupa que esta mal. El tester dice que algo pasa porque no da lo que tiene que dar, el que debe paliar eso es el depurador (Pruebas de regresion).
La **labor del tester** es le preguntan si hay error y este responde SI o NO.

CLASIFICACION DE LAS PRUEBAS

En función del OBJETIVO (VERIFICAR O VALIDAR)

Verificar: Ver si algo falla o no
Validar: Lo ha pedido el cliente?

Hay que tratar con las dos.

NIVELES de pruebas

En funcion en el instante de tiempo del desarrollo.

Cronologicamente: Unitarias -> Integración -> Sistema -> Aceptación.

El objetivo de cada uno de los niveles: Encontrar DEFECTOS blablabla excepto ACEPTACION que son las expectativas del cliente.

PRUEBAS ESTATICAS - DINAMICAS

Objetivo es detectar DEFECTOS.

SOLO VAMOS A TRABAJAR LAS DE BLANCO (DINÁMICAS)
Si el tester puede decir si hay errores ejecutando el código, son pruebas DINAMICAS.

Podemos detectar errores desde el inicio(sin código), puedo prevenir(Estáticas).

CASOS DE PRUEBA

El dato (datos) concreto de entrada + resultado concreto esperado.
¿como?
Proceso de diseño
Automatizar la ejecución del diseño
¿cual es el algoritmo de cada una de las ejecutar de caso de prueba?
Establecer precondiciones.(usuario duplicado, precondición: crear dos usuarios iguales)
Proporcionar datos de entrada
Observar salida (resultado real)
Compara entre datos de entradas y un resultados.

Comportamiento de programa = CASOS DE PRUEBA

El rollo que me suelta el usuario es el conjunto de comportamiento del programa ’S’

La implementación se comportará de la forma ‘P’

’S’ y ‘P’ deben ser idénticos. Pero es muy probable que esto no ocurra.
El tester ’T' se encuentra con este panorama y este elige unas entradas de comportamiento, el tester también puede hacerlo mal y puede hacer pruebas que ni se han pedido.

La mejor zona del tester es la zona 1.
Como puede hacer la maxima.. conseguir el 2 tambien, CASOS DE PRUEBA.
Si aplico todos los diseño de casos de prueba abarcaré el mayor número de errores.

EJECUCION CASOS DE PRUEBA

Automatizar CP, crear un programa, lo ejecute, anote el resultado real y compare para ver si hay fallo o no.

CONSTRUCCION DEL PROYECTO(build)

Proceso o conjunto de tareas que uno debe hacer para pasar de A (situacion inicial) a ese programa en ejecucion B.

Para ello hay que hacer una secuencia de tareas Build script.

Maven nos da tres Build scripts (Ciclo de vida): Secuencia de tareas que deben llevarse a cabo en el BUILD SCRIPT.

Una Fase es una declaración de intenciones (una linea del build script)

Los programas que asocio a cada una de las fases, se llaman GOALS.

Los goals nunca van sueltos, ese conjunto de goals van en un plugin.

ARTEFACTO == fichero.

Los artefactos que utilizan maven o son resultado del proyecto son definidas por coordenadas.
groupID
artefactID: nombre del fichero.
packaging: se puede omitir. Extension .jar
version
Se separa con ':'

Maven almacena todo en repositorios(remotos y locales).

La primera vez se nos creará un directorio oculto en home que se llamará m2.
No puede haber dos coordenadas iguales.

Maven siempre usa un fichero especial llamado **POM**

4 zonas:

Coordenadas: usaremos groupID artifactID version package
Propiedades: properties
dependencias: dependencies (dejaremos todos los ficheros que serán las librerías del programa)
Proceso de construcción: build (enchufar o desenchufar las goals y los plugins)

Todos lo proyectos **maven** tienen la misma forma.

Src
- main
- - java (codigo fuente)
- - resources (ficheros adicionales)
- test
- - java (código fuente de los test)
- - resources

pom.xml
Target (directorios de artefactos

Sesion 1

Fases(hay 23)

Compile (para enchufar algo que compile)
(lo de las diapositivas)

Por defecto MVN añade determinados plugins y determinadas goals en unas fases

Si ejecuto la faseX va desde la 1 a la X
Si ejecuto la goalX ejecuto únicamente esa goal.