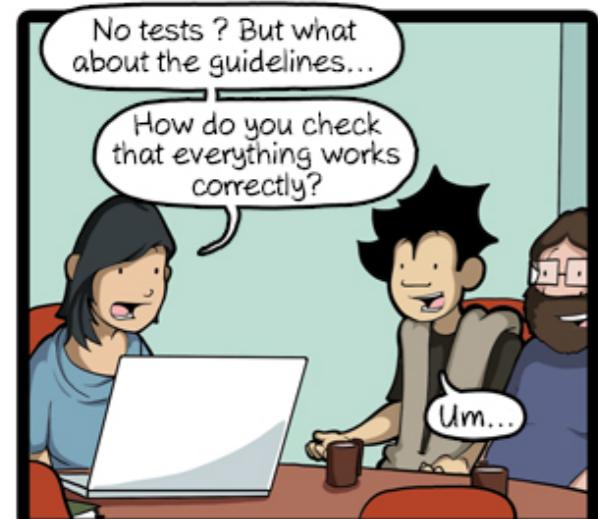
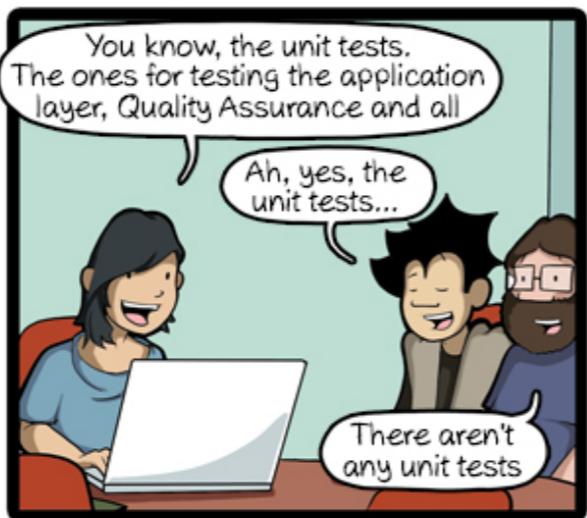
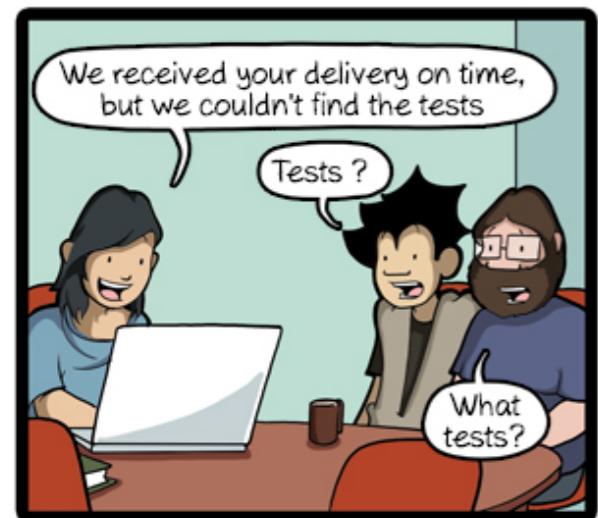


PPSS PLANIFICACIÓN Y PRUEBAS DE SISTEMAS SOFTWARE

Curso 2024-25

Sesión S01: Pruebas del software



CommitStrip.com

Pruebas: qué son, por qué y para qué probamos

Actividades del proceso de pruebas

Principios en los que se basan las pruebas

Pruebas y comportamiento

Construcción de software y pruebas

Vamos al laboratorio...

DEFINICIÓN DEL PROCESO DE PRUEBAS

"Testing is the process of executing a program with the intent of **finding errors**. If our goal is to show the absence of errors, we will discover fewer of them. If our goal is to show the presence of errors, we will discover a large number of them"

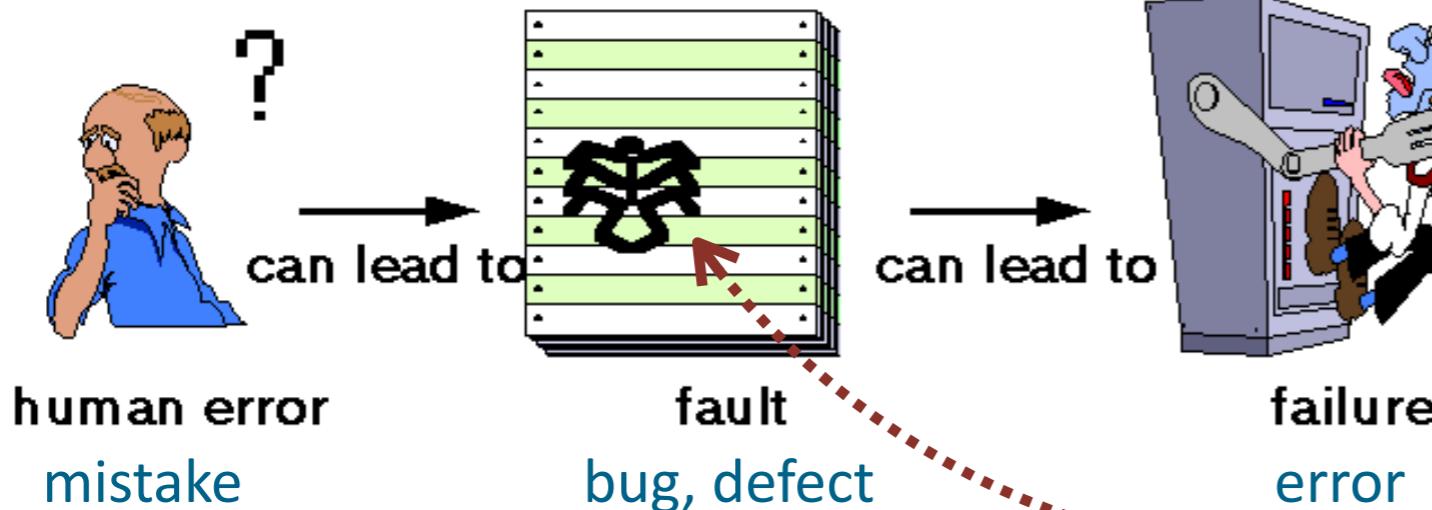
Glenford J. Myers (1979)

“Las pruebas son un conjunto de actividades conducentes a conseguir alguno de estos objetivos:

- * **Encontrar defectos**
- * **Evaluar el nivel de calidad del software**
- * **Obtener información para la toma de decisiones**
- * **Prevenir defectos**

(ISQTB Foundation Level Syllabus 4.0 -2024)

HAY MUCHOS TIPOS DE "ERRORES"!!

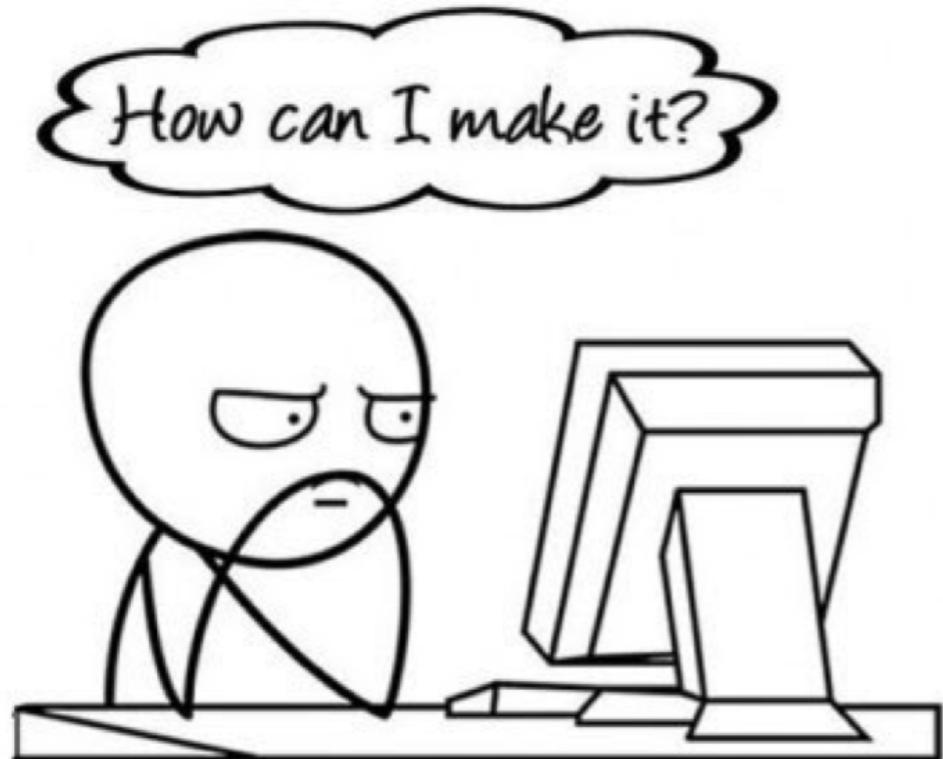


Las pruebas muestran la **PRESENCIA de defectos**. Usaremos técnicas dinámicas para evidenciar los defectos en el código



DEFINICIÓN DEL PROCESO DE PRUEBAS

P
P
S
DEVELOPER



TESTER



A LOS USUARIOS NO LES GUSTAN LOS ERRORES!!



Uno de los objetivos a cumplir para que un proyecto tenga éxito es que el software **satisfaga las expectativas del cliente**

Si las expectativas del cliente no se satisfacen, éste se sentirá justificadamente agraviado

Una prueba tiene EXITO cuando revela la **PRESENCIA de defectos**.

Además, no es suficiente con encontrar defectos, el programa debe satisfacer las necesidades y **expectativas del cliente**)



P

S

S

IMPORTANCIA DE LAS PRUEBAS

No podemos omitir las pruebas si queremos desarrollar un proyecto con ÉXITO

P

¿por qué probamos?

- Necesitamos realizar pruebas porque somos falibles (cometemos errores)
- Durante el desarrollo, aproximadamente un 30-40% de las actividades realizadas están relacionadas con las pruebas

¿para qué probamos?

- Es FUNDAMENTAL realizar un BUEN proceso de pruebas si queremos que nuestro proyecto tenga ÉXITO (se entregue a tiempo, con el coste previsto y satisfaga las expectativas del cliente)

la siguiente cuestión es: ¿cómo lo haremos?

- hay dos grandes grupos de técnicas de pruebas:
 - ESTÁTICAS (no requieren la ejecución de código para detectar defectos. También pueden prevenirlos)
 - DINÁMICAS (requieren ejecutar el código para detectar defectos)



Nos centraremos exclusivamente en las pruebas **DINÁMICAS**

P ACTIVIDADES DEL PROCESO DE PRUEBAS

S SEGÚN ISQTB FOUNDATION LEVEL SYLLABUS (2011)

1 PLANIFICACIÓN y control de las pruebas

Definimos los objetivos de las pruebas, y en todo momento tenemos que asegurarnos de que cumplimos con esos objetivos (p.ej. queremos realizar pruebas sobre el 95% de código)

2 DISEÑO de las pruebas

Es el proceso MÁS IMPORTANTE, si queremos efectivamente cumplir los objetivos marcados. Básicamente consiste en decidir con qué datos de entrada concretos vamos a probar el código, de forma que seamos capaces de detectar el máximo número de errores posibles, en el menor tiempo posible

3 IMPLEMENTACIÓN y ejecución de las pruebas

Creamos código, para probar nuestro código!!



...No, no nos hemos vuelto locos. La idea es que podemos ejecutar las pruebas pulsando un botón, en lugar de hacerlo de forma "manual". Lógicamente, hay que prestarle mucha atención al código de pruebas para que efectivamente nos ayude a conseguir nuestro objetivo

AUTOMATIZACIÓN

4 EVALUACIÓN del proceso de pruebas y emitir un informe

Aplicamos las métricas adecuadas para comprobar si hemos alcanzado los objetivos de pruebas planificados

PRINCIPIOS FUNDAMENTALES DE LAS PRUEBAS

<https://www.boxuk.com/insight/the-seven-principles-of-testing/>

Las pruebas muestran la **PRESENCIA de defectos** (no pueden demostrar la ausencia de los mismos. Si no se encuentra un defecto, no significa que no los haya)

Las pruebas **exhaustivas** son **IMPOSIBLES**

La paradoja del pesticida (los tests deben **revisarse** regularmente para ejercitar diferentes partes del programa)

Clustering de defectos (normalmente los defectos se “concentran” en un reducido número de módulos o componentes del programa)

Las pruebas son **dependientes del contexto** (no es lo mismo probar un sistema de comercio electrónico, o el del lanzamiento de un cohete espacial)

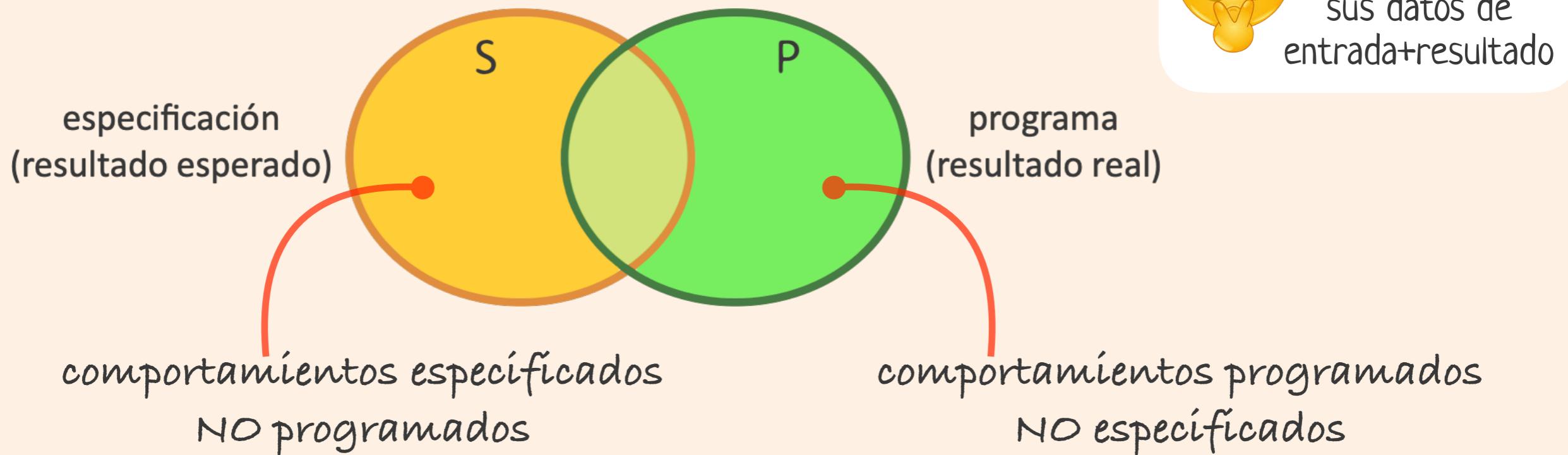
Hay que probar **TAN PRONTO** como sea posible (el coste de reparar un defecto es directamente proporcional al tiempo que transcurre desde que se incurre en él hasta que se descubre)

La **falacia de la ausencia de errores** (no es suficiente el encontrar defectos, el programa debe satisfacer las necesidades y expectativas del cliente)

PSPRUEBAS Y COMPORTAMIENTO

S y P deberían ser idénticos!!!

- Las pruebas conciernen fundamentalmente al **comportamiento** del elemento a probar: ¿qué debería hacer aquello que estoy probando?
- Supongamos un universo de comportamientos de programa. Dado un programa (código) y su especificación, consideraremos:
 - el conjunto S de comportamientos especificados para dicho programa
 - el conjunto P de comportamientos programados



Estos son los problemas con los que se enfrenta un tester!!!

COMPORTAMIENTOS ESPECIFICADOS, PROGRAMADOS, Y PROBADOS

P

- O Incluyamos el conjunto T de comportamientos probados a la figura anterior:

P

especificación
(resultado esperado)

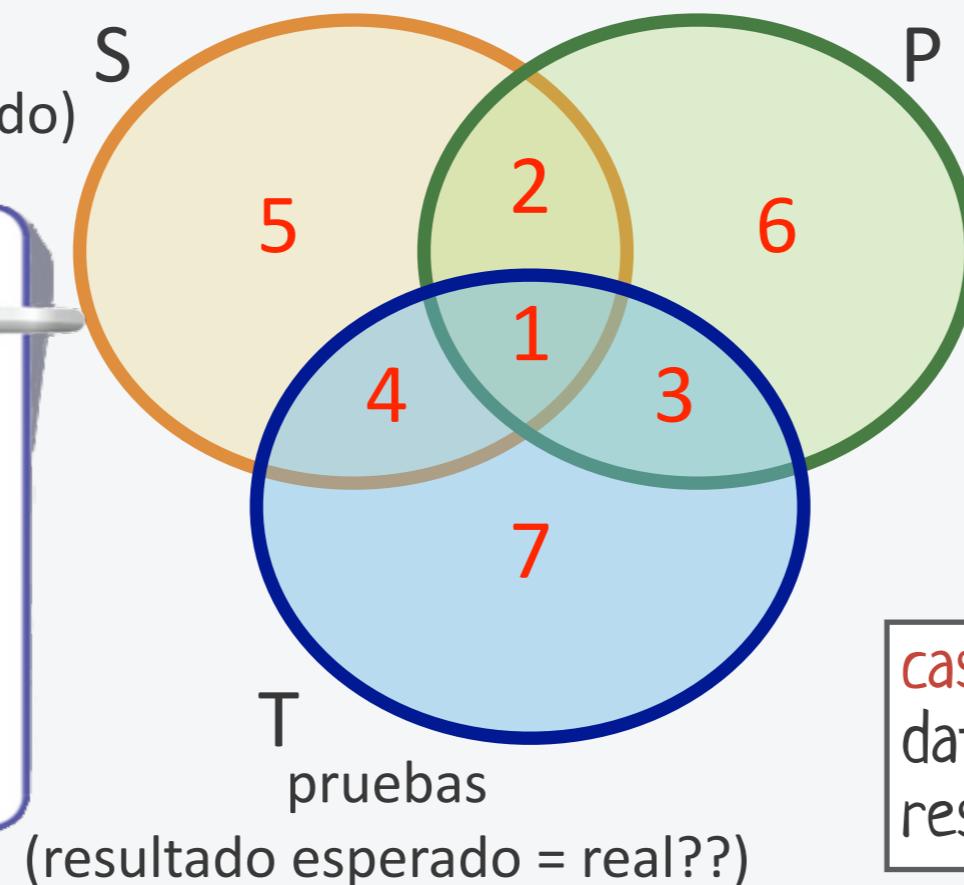
S

Trabajaremos con
técnicas de pruebas
DINÁMICAS.

Una prueba **DINÁMICA**
es aquella que requiere
EJECUTAR el código para
detectar la presencia de
defectos

programa
(resultado real)

Cada una de estas
regiones es importante



Indica qué representan las regiones:

1 2+5

2 1+4

3 3+7

4 2+6

5 1+3

6 4+7

caso de prueba =
datos concretos de entrada +
resultado esperado



- O ¿Qué puede hacer un tester para conseguir que la región 1 sea lo más grande posible?

→ Identificar un conjunto de **casos de prueba** utilizando algún método de **DISEÑO** de pruebas (el objetivo es encontrar el máximo número posible de defectos con el mínimo número de casos de prueba)

EFFECTIVIDAD

EFICIENCIA

PROCESOS DE DISEÑO Y AUTOMATIZACIÓN DE PRUEBAS

El proceso de **DISEÑO** consiste en seleccionar un conjunto de comportamientos a probar

A partir del diseño, necesitamos verificar si dichos comportamientos especificados coinciden con los programados

Dicha verificación, **NECESARIAMENTE** debe realizarse de forma **AUTOMÁTICA**

Cada fila de la tabla representa una prueba (test)

→ TABLA DE CASOS DE PRUEBA

ID	d1	d2	...	Expected
C1				output1
C2				output2
...				...

ID	d1	d2	...	Expected	Real	Informe
C1				output1	R1	✓
C2				output2	R2	fail
...				...		

Se obtienen de forma automática

Vamos a integrar la ejecución de los tests en el proceso de **construcción del sistema**. La construcción del sistema es una actividad que SIEMPRE va a estar presente en el proceso de desarrollo de un proyecto software, por lo que es importante entender en qué consiste dicho proceso

1. Compilamos el código a probar
2. Compilamos el código de los tests
3. Ejecutamos los tests
4. Mostramos el informe de los tests

P

CONSTRUCCIÓN DEL SISTEMA

S

Las pruebas formarán parte
NECESARIAMENTE
de nuestro proceso de construcción

P

O ¿Qué es una construcción del sistema (**build**)?

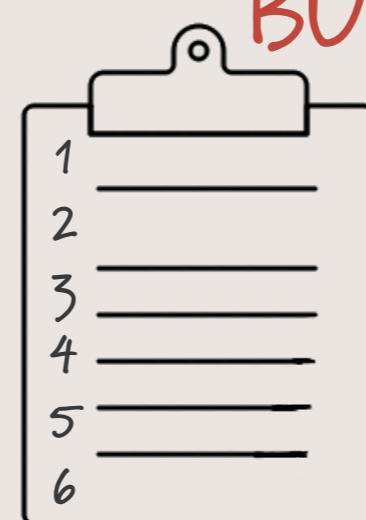
- Es el proceso realizado para "reunir" todo el código fuente ("the process for putting source code together") y verificar que dicho software funciona como una unidad cohesiva.
- El proceso de construcción de un sistema está formado por una **secuencia de acciones** definidas en uno o más "build scripts". Un **build script** puede consistir en una secuencia de compilación, pruebas, empaquetado y despliegue (entre otras cosas)

O Las herramientas de construcción del sistema nos permiten automatizar el proceso de construcción a partir de build scripts

O Ejemplos de herramientas utilizadas para automatizar las construcciones del sistema:

- Make: para lenguaje C
- Ant, Maven, Graddle: para lenguaje Java

Nosotros usaremos
MAVEN !!!



BUILD SCRIPT

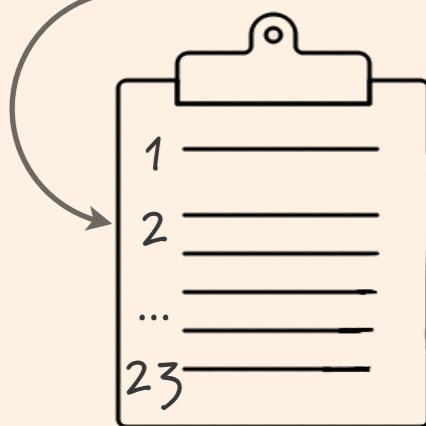
Contiene la **SECUENCIA** de acciones que se ejecutarán de forma automática (pulsando un botón)

MAVEN HERRAMIENTA PARA AUTOMATIZAR LA CONSTRUCCIÓN DE PROGRAMAS Java

Maven estandariza la secuencia de procesos lógicos (FASES) de un build script. A dicha secuencia la denomina CICLO DE VIDA. Maven proporciona 3 ciclos de vida.

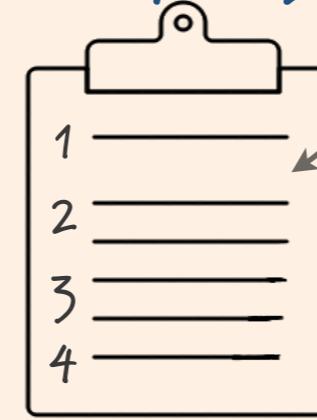
Cada fase sólo pertenece a un único ciclo de vida.

ciclo de vida por
DEFECTO (23 fases)



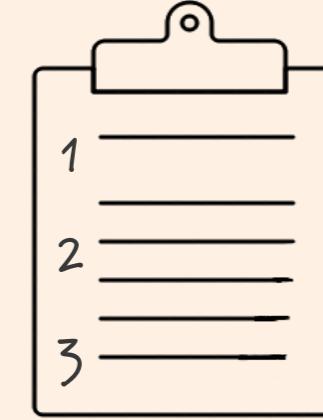
Cada ciclo de vida tiene unas **goals** asociadas por **defecto** a **ALGUNA** de sus fases

ciclo de vida **SITE**
(4 fases)

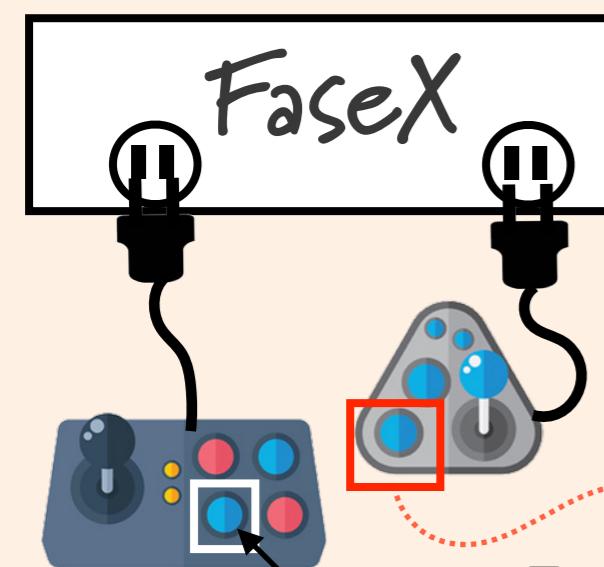


Un ciclo de vida **EJECUTA** siempre las fases de forma **SECUENCIAL** a través de sus **goals** asociadas

ciclo de vida **CLEAN**
(3 fases)



EJEMPLOS de fases: compile, test, package, clean, install, deploy...

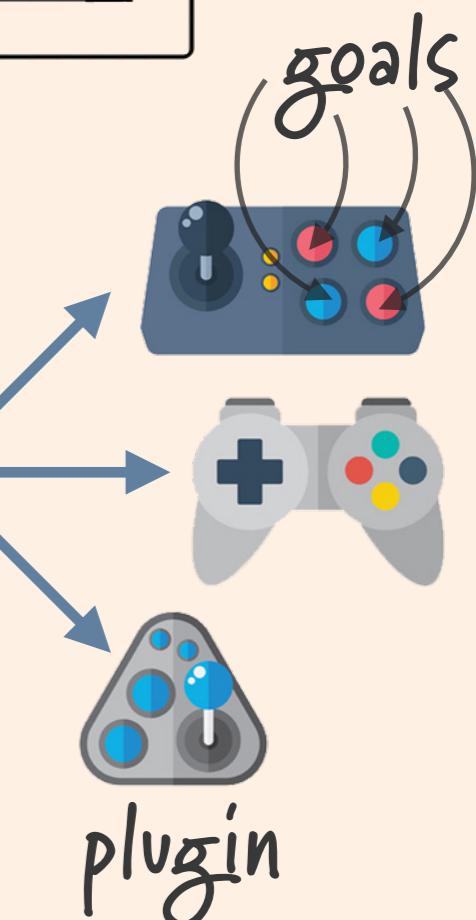


Una FASE PUEDE tener asociadas una o más **GOALS**. una goal es una ACCIÓN EJECUTABLE

Una goal SIEMPRE pertenece a algún **PLUGIN**

esta asociación se configura en el fichero pom.xml

En este ejemplo la fase "FaseX" tiene asociadas las goals "pluginP1:goalA" (goalA pertenece PluginP1), y "pluginP2: goalB"



ESTRUCTURA DE UN PROYECTO MAVEN



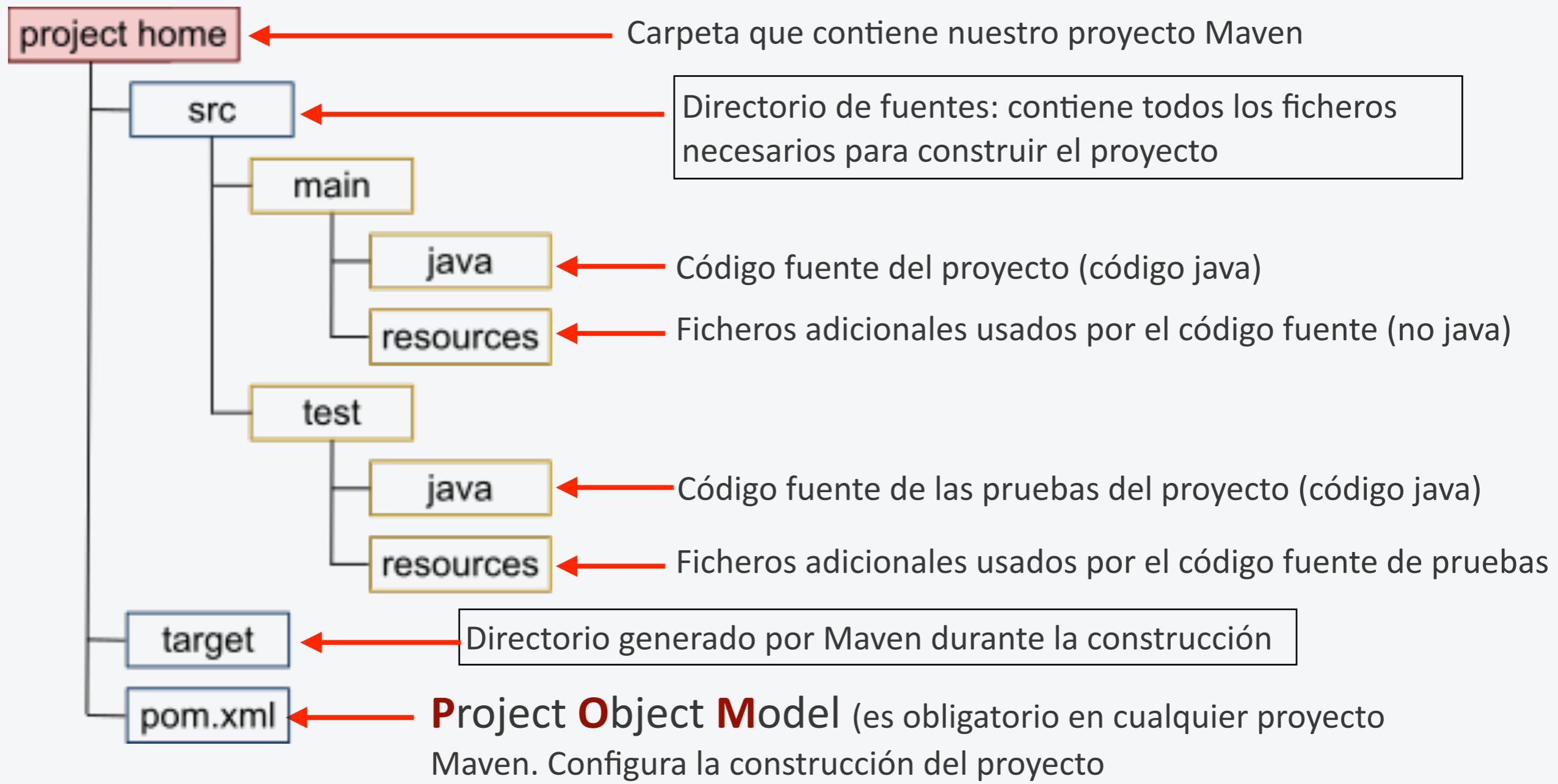
P

Maven estandariza la estructura de directorios de cualquier proyecto java. Un proyecto Maven no tiene por qué tener todos los directorios de la estructura estándar. Por ejemplo,

P

si no necesitamos ningún fichero adicional al código podemos omitir la carpeta src/main/resources

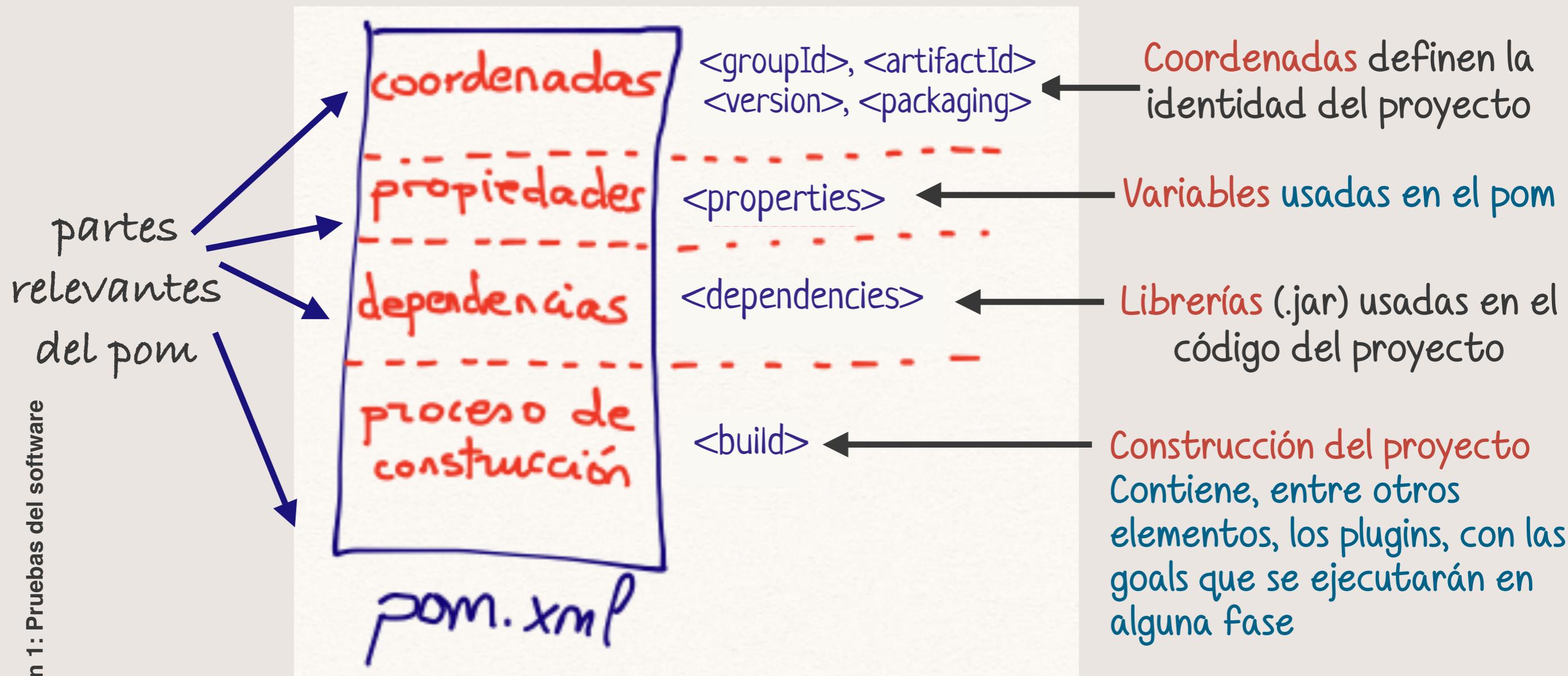
Mostramos PARTE de la estructura de directorios de Maven



P INFORMACIÓN RELEVANTE DE LA CONFIGURACIÓN

TODOS los proyectos Maven tienen un fichero **pom.xml** en su directorio raíz

P En el fichero **pom.xml** configuraremos, de forma declarativa, el build script utilizado por maven para construir el proyecto. Para ello utilizamos diferentes etiquetas xml



ARTEFACTOS MAVEN

P



Un artefacto Maven es un FICHERO identificado con **coordenadas** y que se almacena en un **repositorio**

COORDENADAS (SINTAXIS) → groupId:artifactId:version

- Un **ARTEFACTO** es cualquier fichero que Maven identifica por sus COORDENADAS, y que Maven descarga, instala o despliega de forma automática
 - Ejemplos. ficheros *.pom, *.jar, *.war, *.ear
- Las **COORDENADAS** de un artefacto se indican como: **groupId:artifactId:version**
 - **groupId**: es el identificador de Grupo (<groupId/>)
 - **artifactId**: es el identificador del artefacto (nombre del archivo) (<artifactId/>)
 - **version**: es la versión del artefacto (<version/>)
 - **packaging**. es el empaquetado del artefacto (<packaging/>). Por defecto su valor es **jar**. No se incluye de forma explícita en las coordenadas (aunque podemos añadirlo al final)
- Los artefactos Maven residen en **REPOSITORIOS**:
 - **remotos** (p.ej. <https://mvnrepository.com/>)
 - **locales** (p.ej. \$HOME/.m2/repository/)
- Las coordenadas de un artefacto, no sólo identifican al fichero correspondiente, sino que también nos permiten **LOCALIZAR** exactamente dónde se encuentra dicho fichero (en un repositorio local o remoto). Por ejemplo:
 - **coordenadas**: org.junit.jupiter:junit-jupiter:5.11.4
 - **fichero** en el disco duro: \$HOME/.m2/repository/org/junit/jupiter/junit-jupiter/5.11.4/junit-jupiter-5.11.4.jar
- Cuando maven necesita usar algún artefacto para construir el proyecto, primero comprueba si el fichero correspondiente está almacenado en el repositorio local, y si no, lo descargar del repositorio remoto

CONFIGURACIÓN POR DEFECTO Y EJECUCIÓN DE MAVEN

Las GOALS que están asociadas por defecto a las fases de **DEFAULT LIFECYCLE** (cuando el empaquetado de nuestro proyecto es jar) son las siguientes:

Fase	plugin : goal	acciones
process-resources	maven-resources-plugin: resources	Copia *.* de /src/main/resources en target
compile	maven-compiler-plugin: compile	Compila *.java de /src/main/java
process-test-resources	maven-resources-plugin: testResources	Copia *.* de /src/test/resources en target
test-compile	maven-compiler-plugin: testCompile	Compila *.java de /src/test/java
test	maven-surefire-plugin: test	Ejecuta los tests unitarios
package	maven-jar-plugin:jar	Empaquaeta *.class + recursos en un jar
install	maven-install-plugin: install	Copia el fichero jar en reposit. local
deploy	maven-deploy-plugin: deploy	Copia el fichero jar en reposit. remoto

Para lanzar el proceso de construcción se utiliza el comando **mvn**:

mvn faseM (Se ejecutan TODAS las goals asociadas a las fases, desde fase1 hasta faseM)

mvn pluginX:goalY (Se ejecuta SÓLO la goal goalY del plugin pluginX)

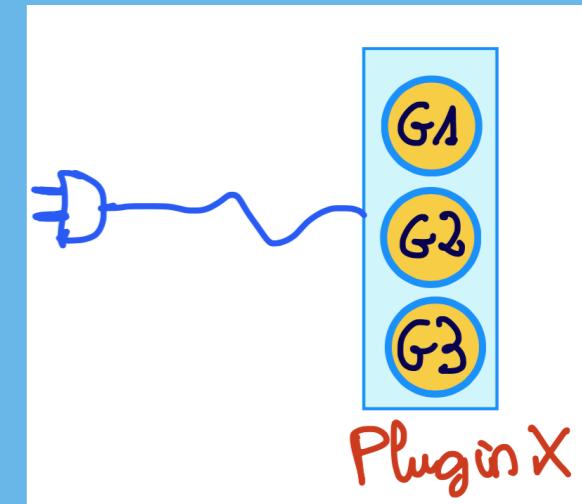
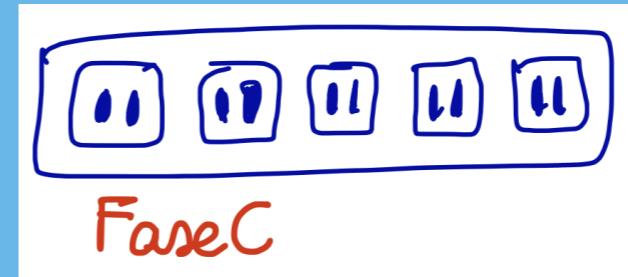
Podemos utilizar la configuración por defecto (secuencia de goals asociadas por defecto a las fases de un ciclo de vida), o bien alterar dicha secuencia (añadir, quitar, modificar goals)

P CONFIGURACIÓN DE LA CONSTRUCCIÓN CON MAVEN

- P ○ Podemos alterar la secuencia de acciones a realizar durante la construcción. Por ejemplo, podemos añadir una determinada acción (goal), incluyendo su plugin en el pom, y activar su ejecución asociándola a una determinada fase

```
<project>
...
<build>
  <plugins>
    <plugin>
      <groupId>...</groupId>
      <artifactId>PluginX<artifactId>
      ...
      <execution>
        <phase>FaseC</phase>
        <goal>G2</goal>
      </execution>
      ...
    </plugin>
  </plugins>
</build>
</project>
```

pom.xml



Ahora, cuando se alcance la FaseC, se ejecutará la goal G2 del PluginX (PluginX:G2)



- También podemos inhibir la ejecución de alguna de las goals asociadas por defecto a una fase

Ahora, cuando se alcance la compile NO se compilarán los fuentes del proyecto



```
<build>
  <plugins>
    <plugin>
      <groupId>...</groupId>
      <artifactId>maven-compiler-plugin<artifactId>
      ...
      <configuration>
        <skip>true</skip>
      </configuration>
      ...
    </plugin>
  </plugins>
</build>
```

pom.xml

Las fases se "recorren" siempre en el mismo orden
comenzando desde la PRIMERA !!!

P



S EJEMPLO

FASES	PLUGIN : GOAL
1 validate	
2 initialize	
3 generate-sources	
4 process-sources	
5 generate-resources	
6 process-resources	resources:resources
7 compile	compiler:compile
8 process-classes	
9 generate-test-sources	
10 process-test-sources	
11 generate-test-resources	
12 process-test-resources	resources:testResources
13 test-compile	compiler:testCompile
14 process-test-classes	
15 test	surefire:test
16 prepare-package	
17 package	jar:jar
18 pre-integration-test	
19 integration-test	
20 post-integration-test	
21 verify	
22 install	install:install
23 deploy	deploy:deploy

mvn test-compile

Ejecuta las goals de
las fases 1...13

Genera los .class de
src/test/java

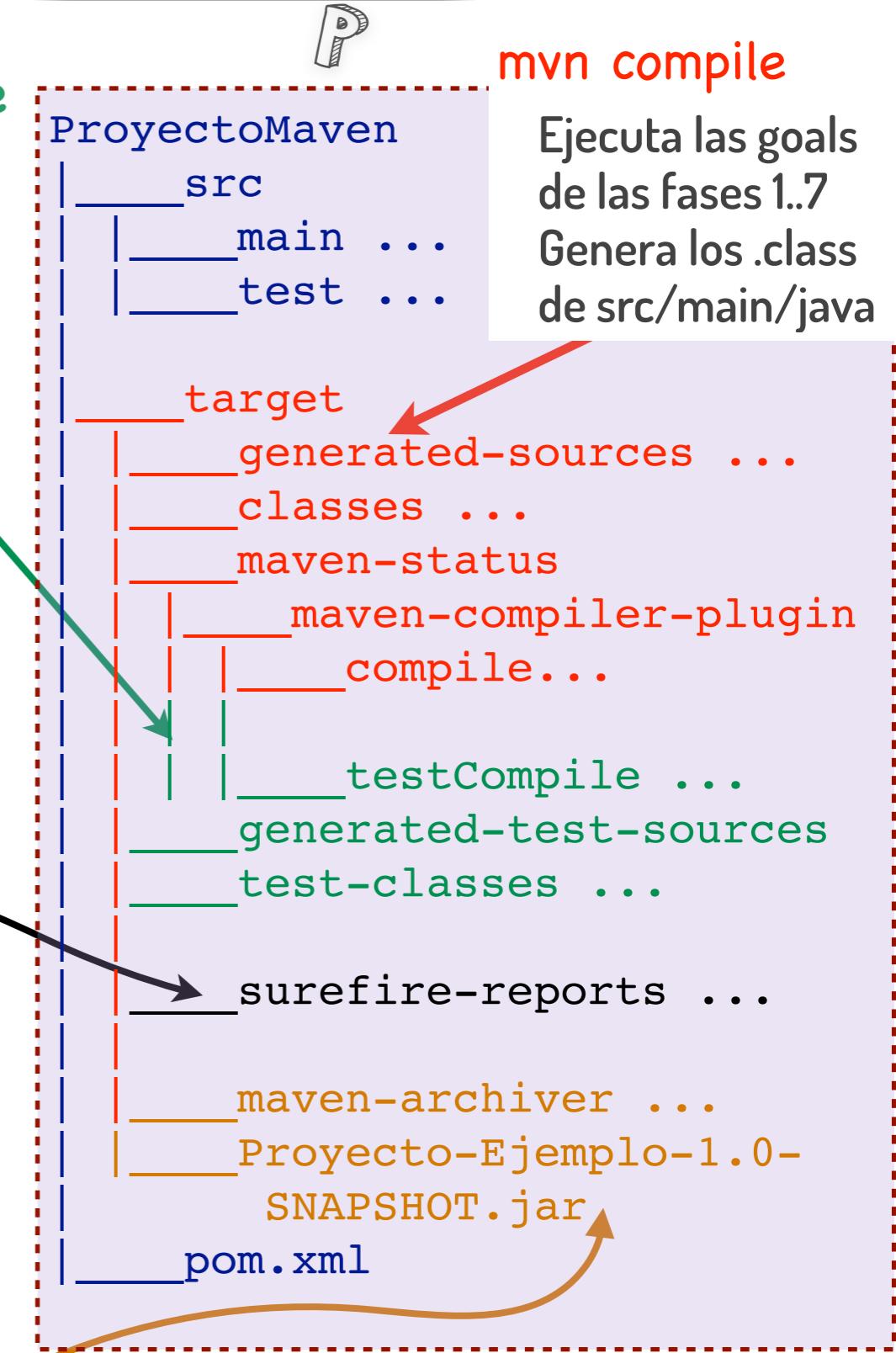
mvn test

Ejecuta las goals de
las fases 1.15

Ejecuta los .class de
src/test/classes y
genera un informe

mvn package

Ejecuta las goals de las fases 1.1.7
Genera el .jar del proyecto



mvn compile

Ejecuta las goals
de las fases 1.7
Genera los .class
de src/main/java

Y AHORA VAMOS AL LABORATORIO...



The screenshot shows the IntelliJ IDEA interface with the following components:

- VISTA DE PROYECTOS (Left Panel):** Shows the project structure under "P00-IntelliJ". The "src/test/java" and "pom.xml" files are highlighted.
- RUN CONFIGURATIONS (Top Bar):** Shows "RUN CONFIGURATIONS" with an arrow pointing to the Maven configuration.
- EDITOR (Central Area):** Displays the content of the "pom.xml" file. The "test" goal is selected in the Maven tool window.
- VENTANA MAVEN (Right Panel):** Shows the Maven tool window with the "Lifecycle" section expanded, displaying various goals like clean, validate, compile, test, package, verify, install, site, and deploy.
- EJECUCIÓN DE LA CONSTRUCCIÓN DEL PROYECTO (Bottom Right):** Shows the build log output in the "Run" tool window, indicating a "BUILD SUCCESS" and a total time of 2.964 seconds.

REFERENCIAS BIBLIOGRÁFICAS

- Software Testing. A craftsman's approach. 5th edition. Paul C. Jorgensen (2021)
 - Capítulo 1. A perspective on testing
- ISTQB. Foundations level syllabus v4.0 (https://www.gasq.org/files/content/ISTQB2/ISTQB_CTFL_Syllabus_v4.0.1.pdf)
- Apache Maven (<https://maven.apache.org/>):
 - [Maven central repository](#)
 - [Maven pom reference](#)
 - [Guide to configuring plugins](#)
 - [Introduction to the build lifecycle](#)