Planificación y Pruebas de Sistemas Software

Curso 2015-16

Sesión 1: Pruebas del software

Pruebas: qué son, por qué y para qué probamos
Principios de las pruebas
Pruebas y depuración
Niveles y técnicas de pruebas
Casos de prueba
Construcción de software y pruebas

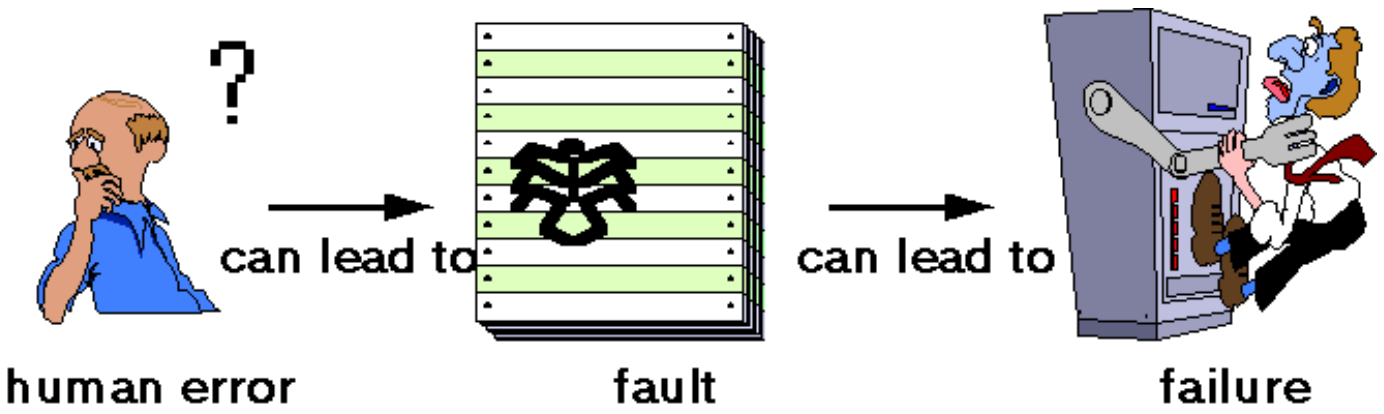
María Isabel Alfonso Galipienso
Universidad de Alicante eli@ua.es

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 (ISQTB Foundation Level Syllabus -2011) :
  - Encontrar defectos
  - Evaluar el nivel de calidad del software
  - Obtener información para la toma de decisiones
  - Prevenir defectos



Importancia de las pruebas

- Uno de los objetivos de éxito del proyecto es que el software satisfaga las expectativas del cliente
  - Si las expectativas del cliente no se satisfacen, éste se sentirá justificadamente agraviado
- Necesitamos realizar pruebas porque somos falibles (cometemos errores)
- Durante el desarrollo, aproximadamente un 30-40% de las actividades están relacionadas con las pruebas
  - Es **FUNDAMENTAL** realizar un **BUEN** proceso de pruebas si queremos que nuestro proyecto tenga ÉXITO (se entrega a tiempo, con el coste previsto y satisface las expectativas del cliente)

Principios fundamentales de las pruebas

- 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 (es impracticable probar todas las combinaciones posibles de entradas de un programa)
- Hay que probar tan pronto como sea posible (el coste de reparar un defecto es inversamente proporcional al tiempo que transcurre desde que se incurre en él hasta que se descubre)
- Clustering de defectos (normalmente los defectos se "concentran" en un reducido número de módulos o componentes del programa)
- La paradoja del pesticida (los tests deben revisarse regularmente para ejercitar diferentes partes del programa)
- Las pruebas son dependientes del contexto (no es lo mismo probar un sistema de comercio electrónico, que el lanzamiento de un cohete espacial)
- La falacia de la ausencia de errores (no es suficiente el encontrar defectos, el programa debe satisfacer las necesidades y expectativas del cliente)

Objetivos de las pruebas

- Fundamentalmente probamos para:
  - Juzgar la calidad del software o en qué grado es aceptable
    - Este proceso se denomina VALIDACIÓN: se trata de ver si el producto desarrollado satisface las expectativas del cliente (comprobamos si lo que estamos construyendo es lo que el cliente quiere y/o necesita)
  - Detectar problemas
    - Este proceso se denomina VERIFICACIÓN: se trata de buscar defectos en el programa que provocarán que éste no funcione correctamente (según lo esperado, de acuerdo con los requerimientos especificados previamente)
- Estos dos procesos son necesarios y complementarios:
  - Un producto puede funcionar correctamente, pero no satisfacer las expectativas del cliente (no es exactamente el producto que quería)
  - En cualquier desarrollo, independientemente del modelo de proceso que se siga, se deben llevar a cabo actividades tanto de verificación como de validación

Testing y SQA

- Diferencias entre proceso y producto:
  - El PROCESO hace referencia a CÓMO hacemos algo
  - El PRODUCTO es el RESULTADO de un proceso
- SQA (Software Quality Assurance) son las actividades conducentes a mejorar el producto mediante la mejora del proceso
- Las pruebas (testing) tienen que ver con SQA (Software Quality Assurance) en tanto que:
  - el proceso de testing es orientado al PRODUCTO, pretende descubrir los DEFECTOS del mismo (un defecto es el resultado de un error humano)
  - el proceso de SQA pretende reducir los errores del PROCESO de desarrollo (que finalmente producirán DEFECTOS en el producto resultante)
- Los DEFECTOS (o faults) pueden categorizarse de diferentes formas:
  - input/output faults, logic faults, computation faults, interface faults, data faults (ver http://www.ctestlabs.org/neoacm/1044_2009.pdf)

Testing y debugging

- Los procesos de pruebas y depuración (debugging) son diferentes
  - El proceso de **pruebas** concluye cuando se DETECTA un fallo (*failure*) del programa (discrepancia entre el resultado esperado y el resultado real), lo cual es síntoma de que hay un defecto en el programa. Los responsables de realizar las pruebas son los **testers**
  - El proceso de **depuración** es una actividad del desarrollo que: encuentra, analiza y elimina la CAUSA del fallo de ejecución (*failure*), es decir, elimina el defecto que provoca el fallo de ejecución. Normalmente la depuración la realizan los **desarrolladores**
- Siempre que depuramos algún defecto, hay que volver a repetir las pruebas (las veces que sea necesario) hasta asegurarnos de que hemos corregido la causa del fallo de ejecución
- Cualquier cambio realizado en el programa requerirá REPETIR las pruebas para asegurarnos de que este cambio no provoca fallos nuevos, ni se introducen nuevos defectos. Este proceso se denomina **regression testing**

Niveles de pruebas

ver ISQTB Foundation Level Syllabus (2011)

- Pruebas de unidades (componentes)
  - Objetivo: encontrar defectos en el código de las unidades probadas
  - Una cuestión fundamental es "**aislar**" nuestra unidad del resto del código (independientemente de la granularidad de lo que definamos como unidad)
- Pruebas de integración
  - Objetivo: encontrar defectos derivados de la **interacción** entre las unidades, que previamente han sido probadas
  - La cuestión fundamental es el "**orden**" en el que vamos a realizar dicha integración
- Pruebas del sistema
  - Objetivo: encontrar defectos derivados del **comportamiento** del software como un **todo**
- Pruebas de aceptación
  - Objetivo: valorar en qué grado el software desarrollado **satisface** las **expectativas** del cliente

Técnicas de pruebas

ver ISQTB Foundation Level Syllabus (2011)

- Pruebas estáticas
  - No requieren ejecutar código para detectar defectos en el software
  - Pueden aplicarse en cualquier momento del desarrollo
  - Ejemplos de defectos que pueden encontrarse: desviación de los estándares establecidos, defectos en los requerimientos, defectos en el diseño, mantenibilidad reducida, especificaciones de interfaces incorrectas…
  - Reducen el coste de reparación de los defectos encontrados, ya que permiten detectarlos de forma temprana (en comparación con las pruebas dinámicas). Se centran en detectar las causas de los fallos de ejecución
- Pruebas dinámicas
  - Requieren ejecutar código para detectar defectos en el software
  - Sólo pueden aplicarse si de dispone del código correspondiente
- En ambos casos, el objetivo es el mismo: identificar defectos. Se trata de técnicas COMPLEMENTARIAS.

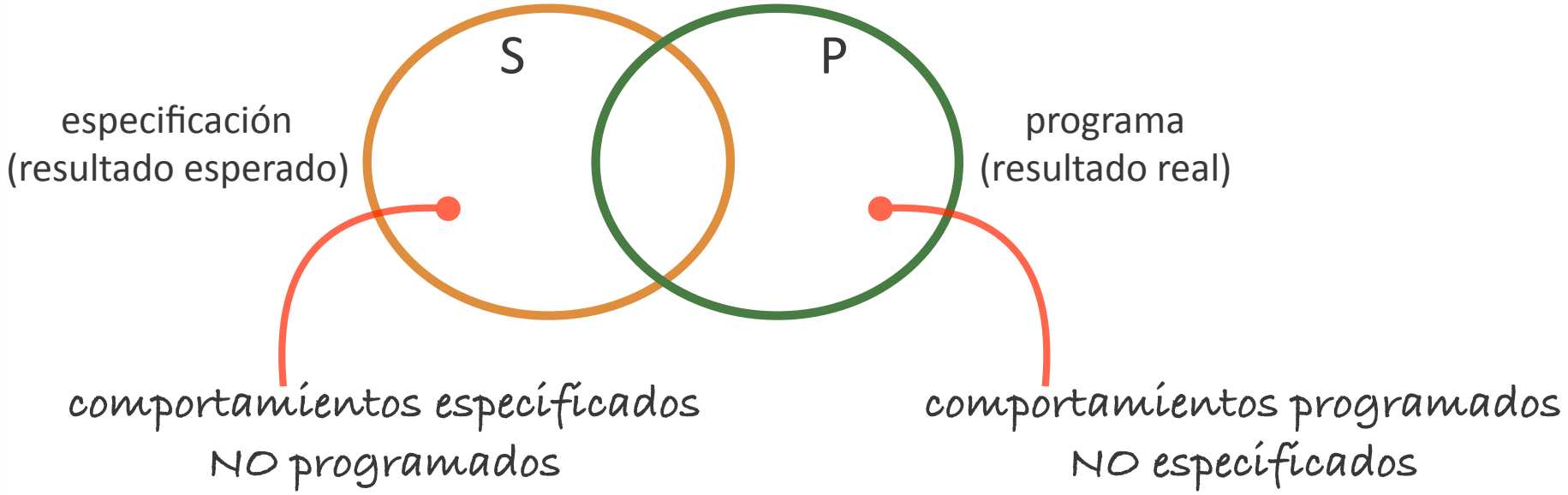
Casos de prueba

- Para realizar pruebas, es ESENCIAL determinar un conjunto de CASOS DE PRUEBA para cada elemento a probar
- Un caso de prueba está formado por:
  - Datos CONCRETOS de entrada del elemento a probar
  - El resultado CONCRETO esperado, dadas las entradas anteriores
- La ejecución de un caso de prueba requiere:
  - Establecer las precondiciones (asunciones sobre lo que es cierto antes de ejecutar el caso de prueba)
  - Proporcionar los datos de entrada + el resultado esperado
  - Observar la salida (resultado real)
  - Comparar el resultado esperado con el resultado real
  - Emitir un informe (para poner de manifiesto si hemos detectado un fallo o no)

1. Piensa en algún ejemplo de caso de prueba
2. Indica algún ejemplo de precondición

Pruebas y comportamiento

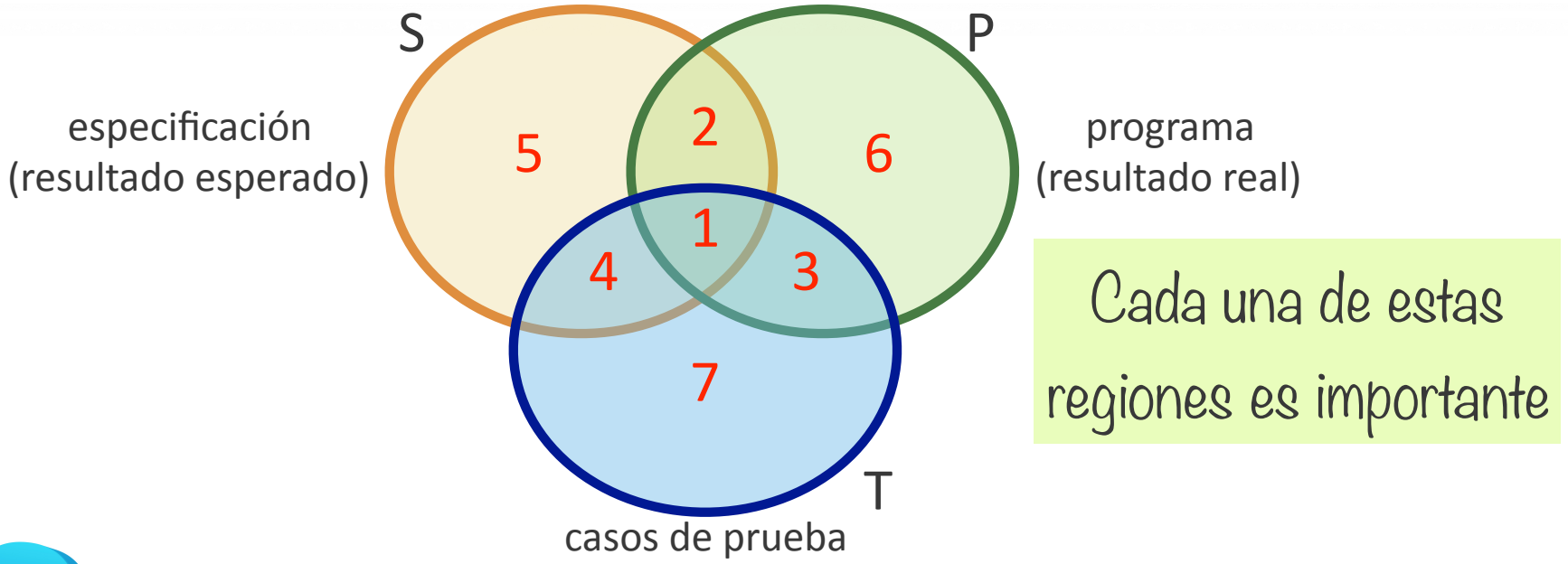
- Las pruebas conciernen fundamentalmente al comportamiento del elemento a probar: ¿qué debería hacer aquello que estoy probando?
- Consideremos un universo de comportamientos de programa. Dado un programa y su especificación, consideremos:
  - el conjunto S de comportamientos especificados para dicho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



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

Comportamientos especificados, programados, y probados

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



Cada una de estas regiones es importante

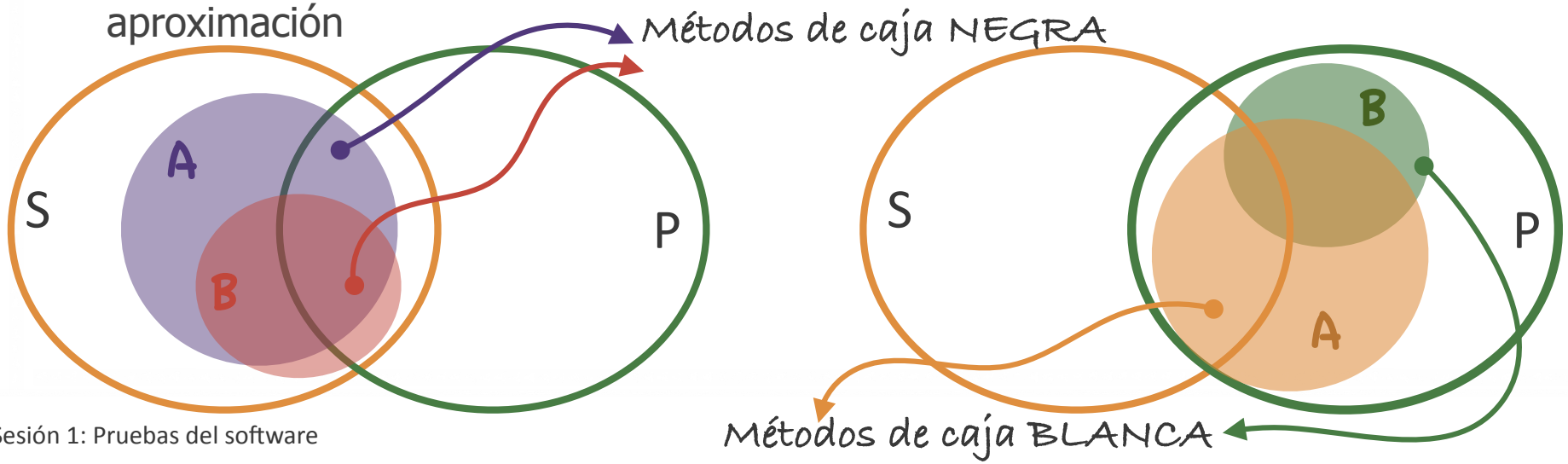
Indica qué representan las regiones:

1 2+5 2 1+4 3 3+7 4 2+6 5 1+3 6 4+7

- ¿Qué puede hacer un tester para conseguir que la región 1 sea lo más grande posible?
  - Identificar el conjunto de casos de prueba utilizando algún método de DISEÑO de pruebas

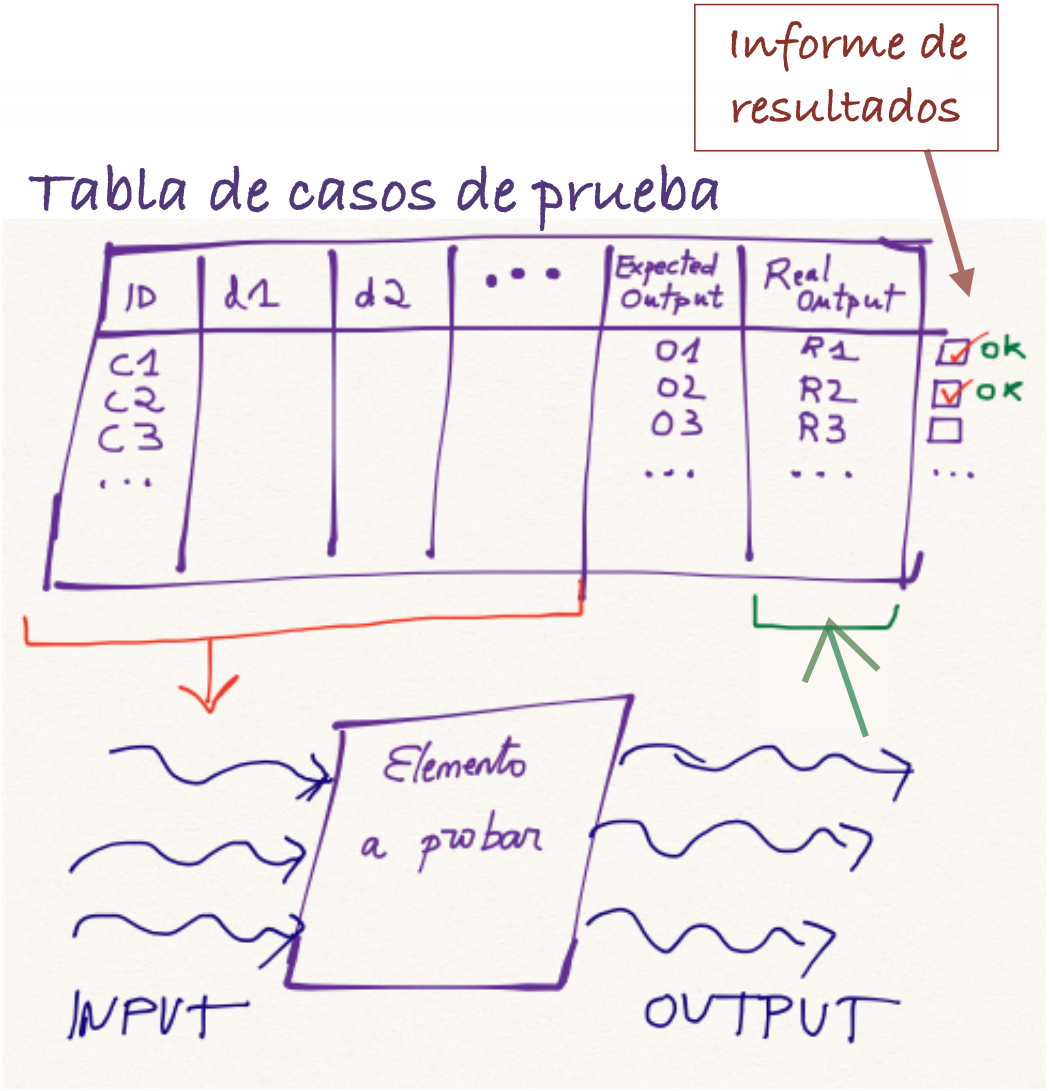
Formas de identificar los casos de prueba

- Fundamentalmente hay dos formas de proceder:
  - ***Functional testing***: aproximación basada en la ESPECIFICACIÓN
    - Cualquier programa puede considerarse como una función que "mapea" valores desde un dominio de entrada a valores en un dominio de salida. El elemento a probar se considera como una "caja negra"
    - Los casos de prueba obtenidos son independientes de la implementación
    - El diseño de los casos de prueba puede realizarse en paralelo o antes de la implementación
  - ***Structural testing***: aproximación basada en la IMPLEMENTACIÓN
    - Utilizamos la implementación para determinar el conjunto de casos de prueba. También se conoce con el nombre de "caja blanca"
    - Es esencial conocer conceptos de teoría de grafos para entender bien esta aproximación



Ejecución de casos de prueba

- Una vez identificados los casos de prueba, podemos **automatizar** su ejecución, para ello utilizaremos diferentes herramientas: JUnit, DbUnit, Selenium, Jmeter,…
- La **ejecución** de los casos de prueba nos permite obtener un "informe" con el resultado de los tests. Por ejemplo, si utilizamos JUnit, este informe será: Pass, Failure, o Error, para cada test
- La ejecución de los casos de prueba forma parte del 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

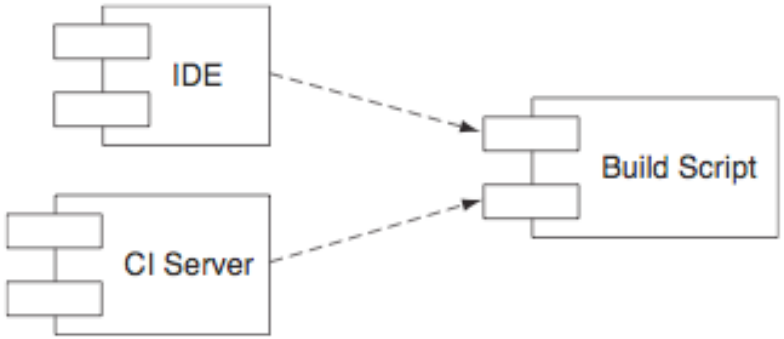


Construcciones del sistema

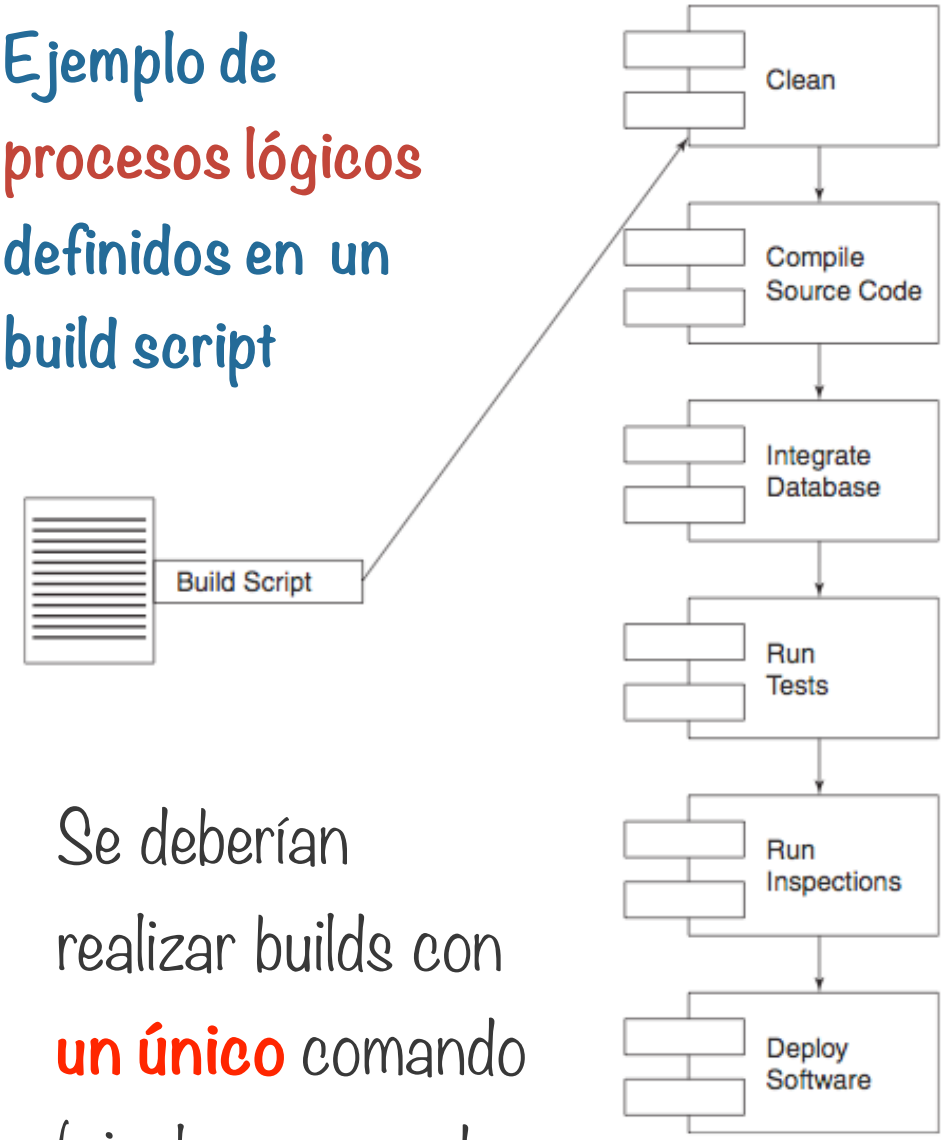
- ¿Qué es una construcción del sistema (build)?
  - Una construcción (build) es mucho más que una compilación del código fuente. Una build puede consistir en una secuencia de compilación, pruebas, empaquetado y despliegue (entre otras cosas). Una construcción es el proceso realizado para "reunir" todo 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*"
  - Las herramientas de construcción del sistema nos permiten "definir" la secuencia de acciones (que puede ser diferente para cada proyecto) para construir el proyecto
- Ejemplos de herramientas utilizadas para automatizar las construcciones del sistema:
  - Make: para lenguaje C
  - Ant, Maven, Graddle: para lenguaje Java

Build scripts

- Un build script contiene la secuencia de procesos lógicos que definen el proceso de construcción de un proyecto
- Deberíamos evitar el acoplamiento de los build scripts con un IDE
  - Un IDE puede depender de un build script, pero un build script NO debería depender de un IDE



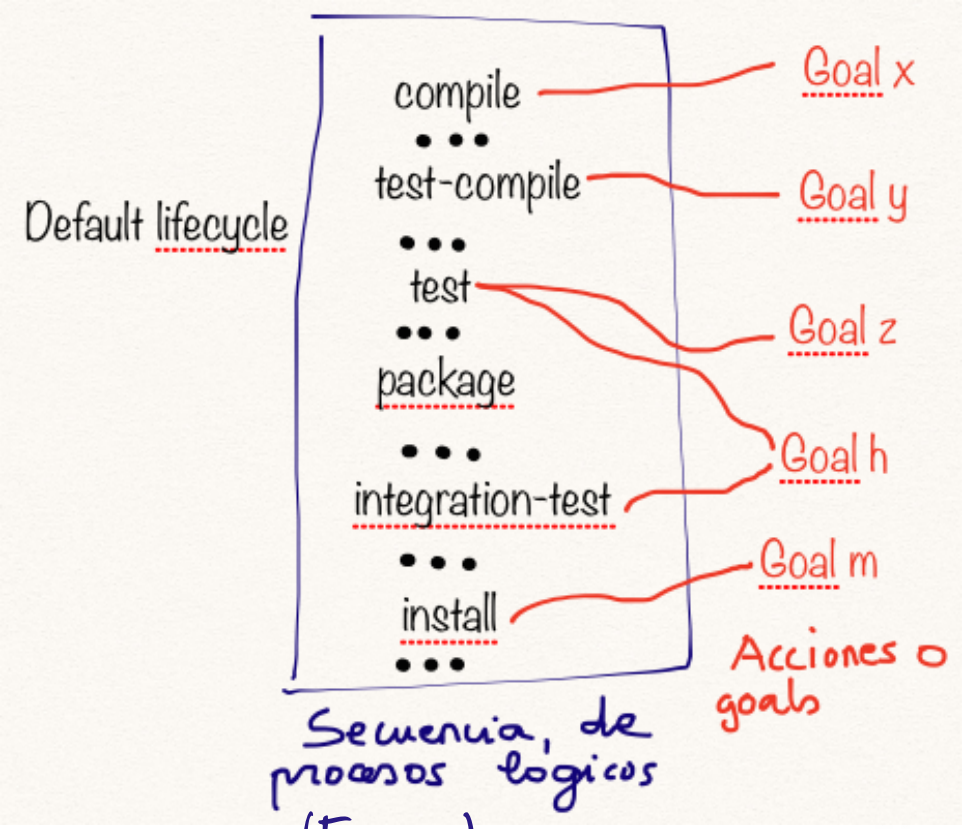
Ejemplo de procesos lógicos definidos en un build script

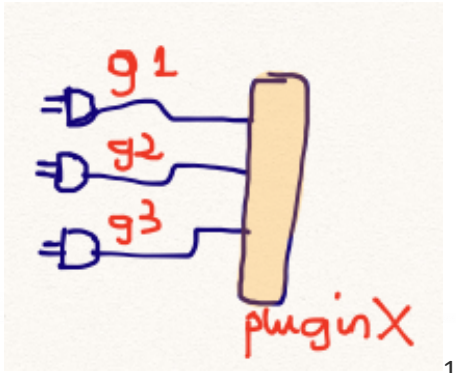


Se deberían realizar builds con **un único** comando (single command builds)

Maven

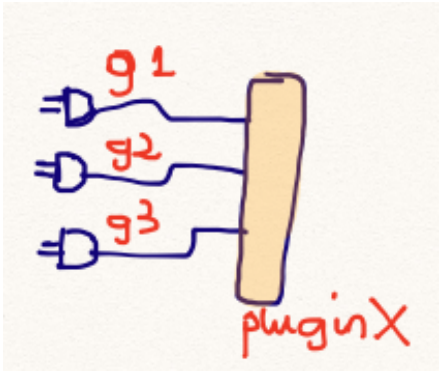
- Maven es una herramienta para automatizar la construcción de programas Java:
  - Maven estandariza la **secuencia** de procesos lógicos de ejecución (fases). Dicha secuencia se denomina ciclo de vida
    - Cada fase tiene asociada por defecto unas acciones (goals). Hay fases que por defecto no tienen asociadas ninguna goal
    - Cada goal está asociada por defecto a una fase. Hay goals que por defecto no están asociadas a ninguna fase.
    - Cada goal está definida dentro de un plugin
  - Para lanzar el proceso de construcción se utiliza el comando **mvn:**
    - mvn faseZ (Se ejecutan TODAS las goals desde fase 1 hasta faseZ)
    - mvn pluginX:g1 (Se ejecuta sólo la goal g1)

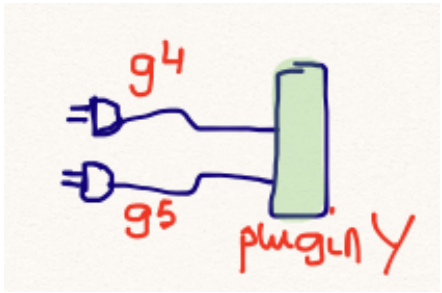


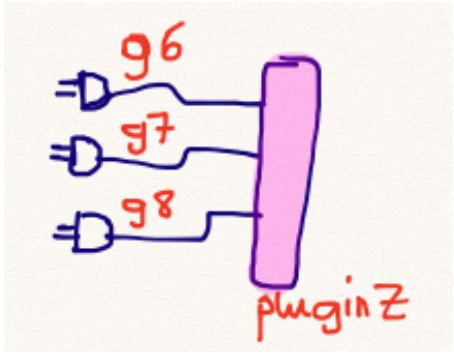


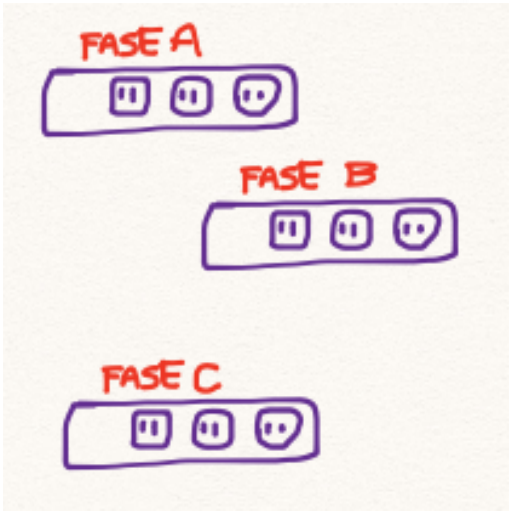
Configuración de la construcción con Maven

- Podemos utilizar la configuración por defecto, o bien configurar nuestro propio proceso de construcción (secuencia de acciones)





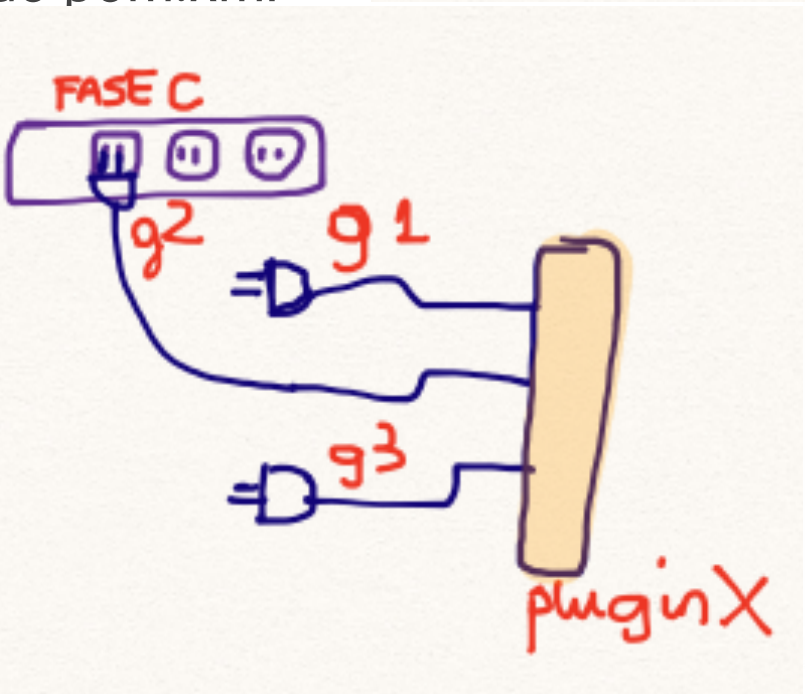




- Para ello utilizamos un fichero denominado pom.xml

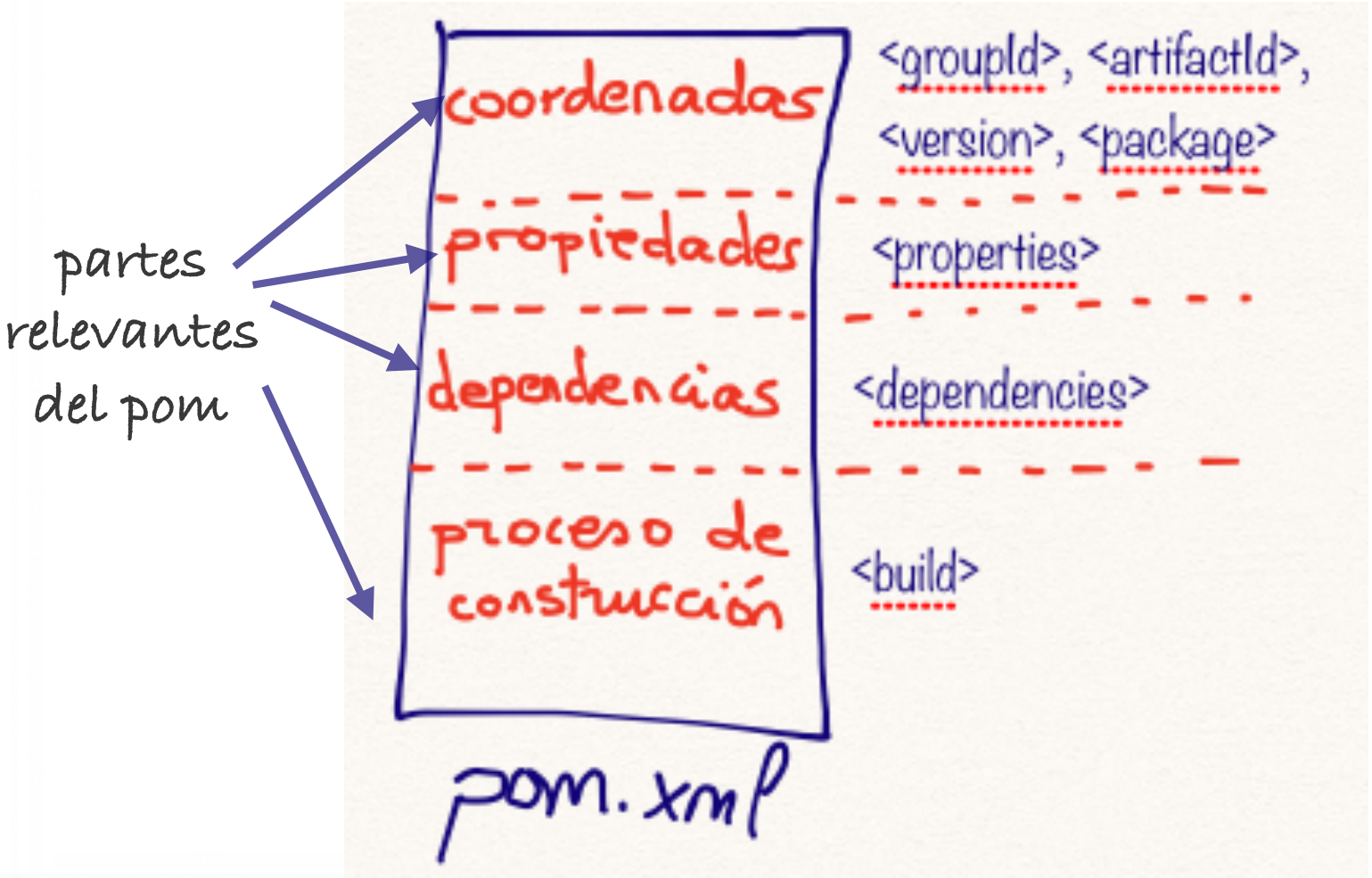
contenido del pom.xml

```
<project>
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>...</groupId>
        <artifactId>PluginX<artifactId>
        ...
        <execution>
          <phase>FaseC</phase>
          <goal>g2</goal>
        </execution>
            ...
</project>
```



Información relevante de la configuración

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



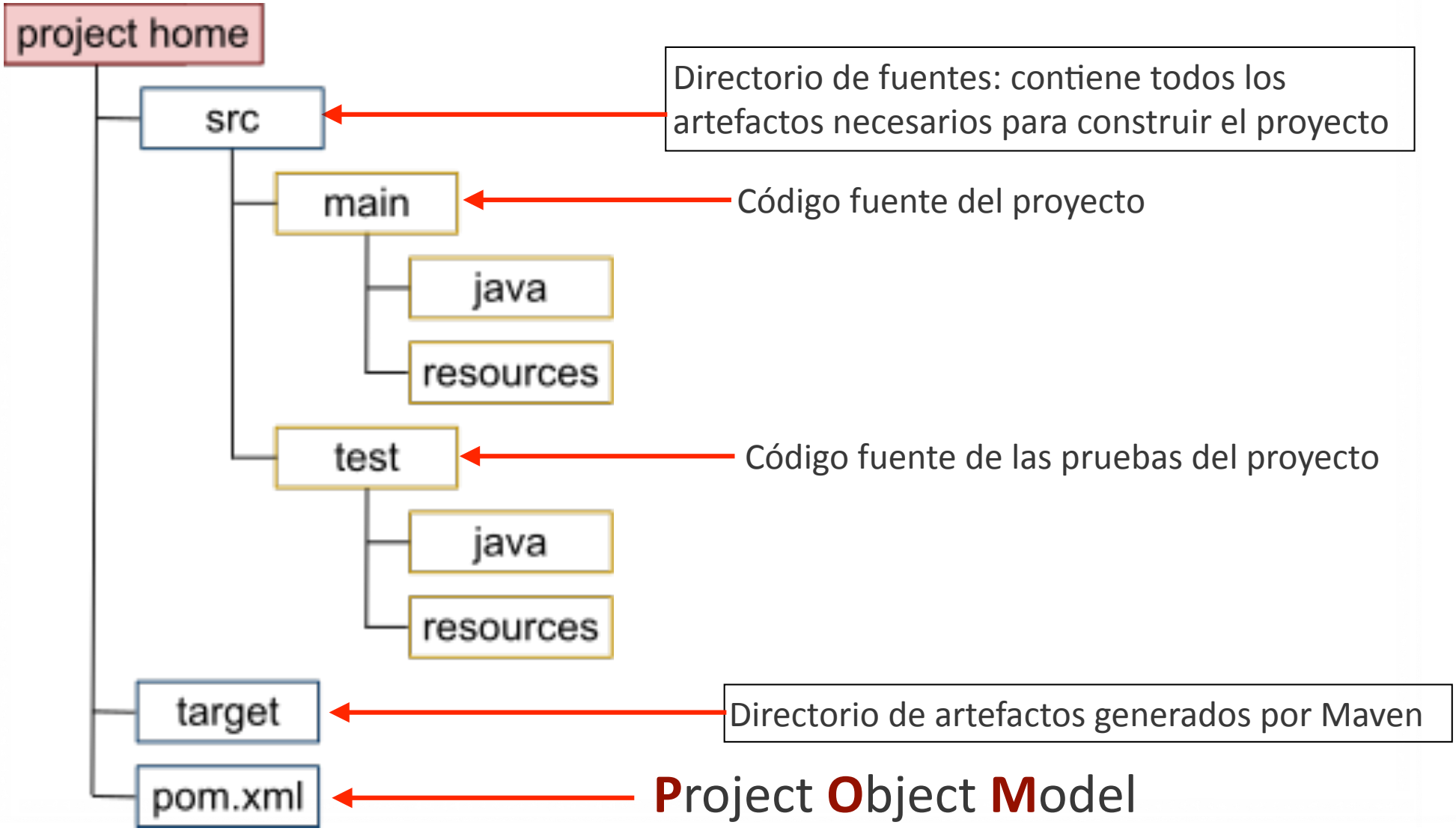
Identificación de los artefactos Maven

`groupId:artifactId:packaging:version`

- Cualquier artefacto utilizado por Maven se identifica por sus COORDENADAS:
  - Identificador de Grupo (groupId)
  - Identificador del artefacto (artifactId)
  - Versión del artefacto (version)
  - Empaquetado del artefacto (package), por defecto el empaquetado es "jar"
- Ejemplos:
  - org.apache.maven.plugins:maven-compiler-plugin:jar:3.2
  - mi.practica:practica7:jar:1.0.SNAPSHOT
  - junit:junit:4.12
- Los artefactos Maven residen en REPOSITORIOS:
  - globales (p.ej. http://mvn.repository.com)
  - locales (p.ej. $HOME/.m2/repository)
  - La identificación del artefacto se utiliza para almacenar y localizar el artefacto físicamente en el repositorio correspondiente
    - P.ej. $HOME/.m2/repository/junit/junit/4.12/junit-4.12.jar

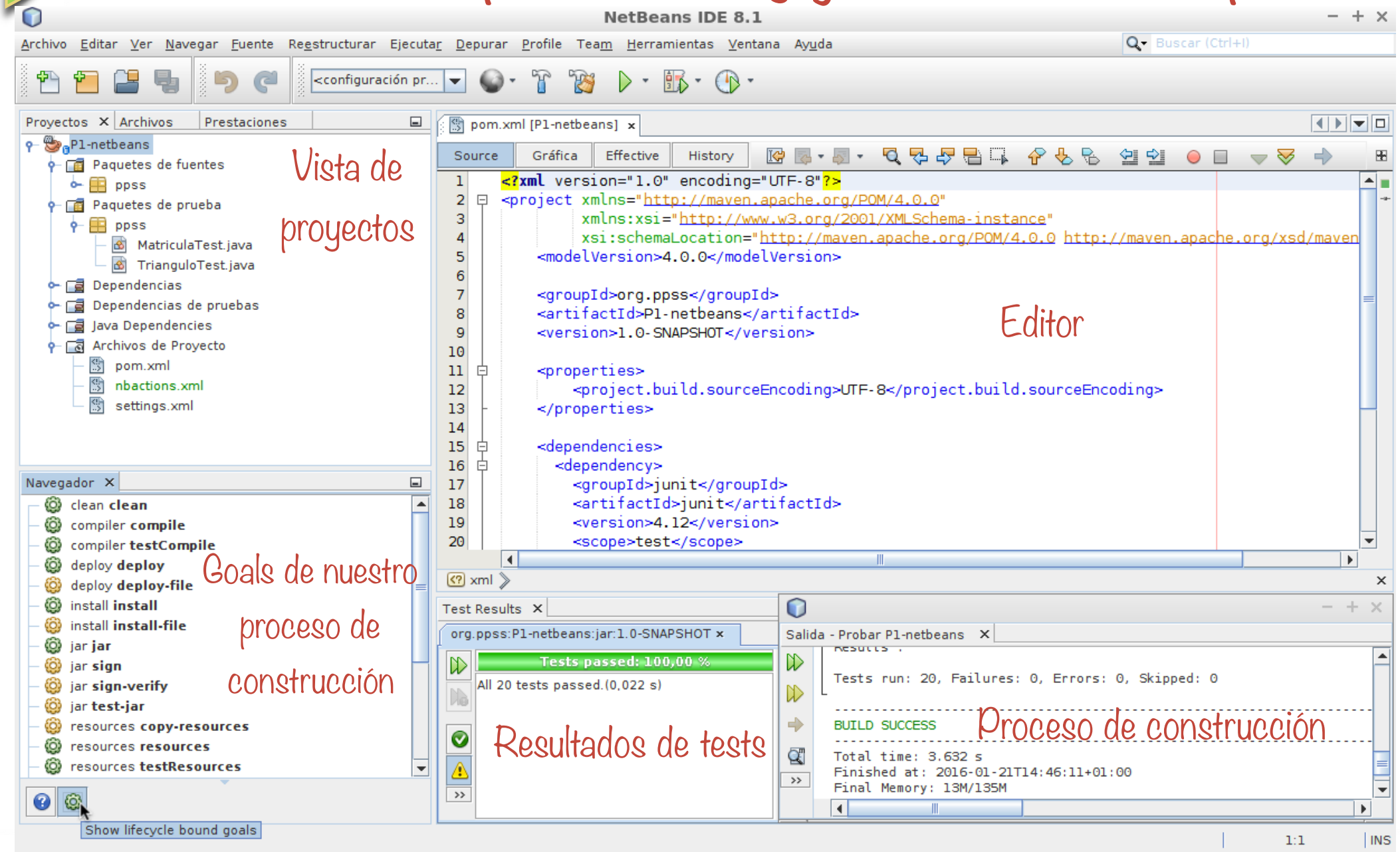
Estructura de un proyecto Maven

- Maven estandariza los elementos de configuración de los proyectos. Un proyecto Maven sigue la siguiente estructura estándar:



Y ahora vamos al laboratorio…

Diseñaremos, implementaremos y ejecutaremos casos de prueba



Referencias bibliográficas

- Software Testing. A craftsman's approach. 4th edition. Paul C. Jorgensen (2014) ISBN-13: 978-1-4556-6068-0
  - Capítulo 1. A perspective on testing
- ISTQB. Foundations level syllabus (http://www.istqb.org/downloads/syllabi/foundation-level-syllabus.html)
- Maven (http://maven.apache.org)