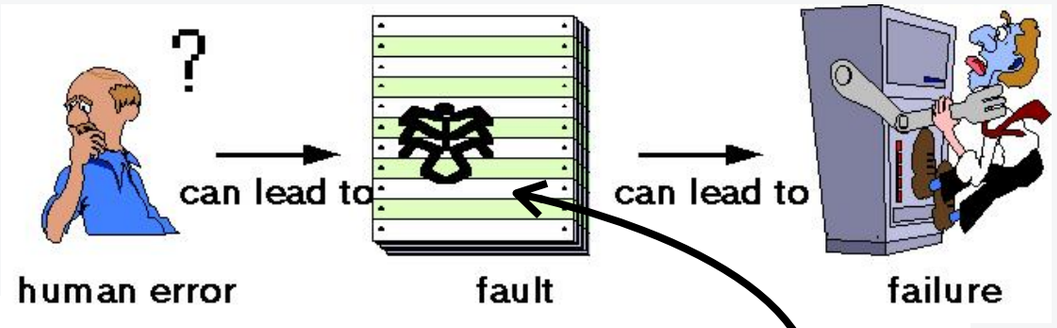
REPASO SESIONES S01..S06

SESIÓN S01



PROBAMOS PORQUE ...

- Cometemos errores

Las pruebas consumen mucho tiempo del desarrollo!!!

CÓMO PROBAMOS??

PROCESO

- PLANIFICACIÓN
- DISEÑO
- AUTOMATIZACIÓN
- EVALUACIÓN

PROBAMOS PARA ...

- Encontrar defectos (VERIFICACIÓN)
- Juzgar si la calidad del sw es aceptable (VALIDACION)
- Prevenir defectos (Pruebas estáticas)
- Toma efectiva de decisiones (Métricas)

TÉCNICAS

Pruebas DINÁMICAS (es necesario ejecutar el código):

- Pruebas UNITARIAS: encuentran DEFECTOS en las unidades. Necesitamos AISLAR nuestro SUT (controlando sus dependencias externas con DOBLES)
  - Diseño:
    - métodos de caja BLANCA (camino básico) (SESIÓN S02)
    - métodos de caja NEGRA (particiones equivalentes) (SESIÓN S04)
  - Automatización
    - Drivers (verific basada en el estado, Usan STUBS para controlar las entradas indirectas de nuestro SUT) (S03) (S05)
    - Drivers (verif. basada en el comportamiento. Usan MOCKS para observar las entradas indirectas y registrar las interacciones de nuestro SUT con sus dependencias externas (S06)

HERRAMIENTAS

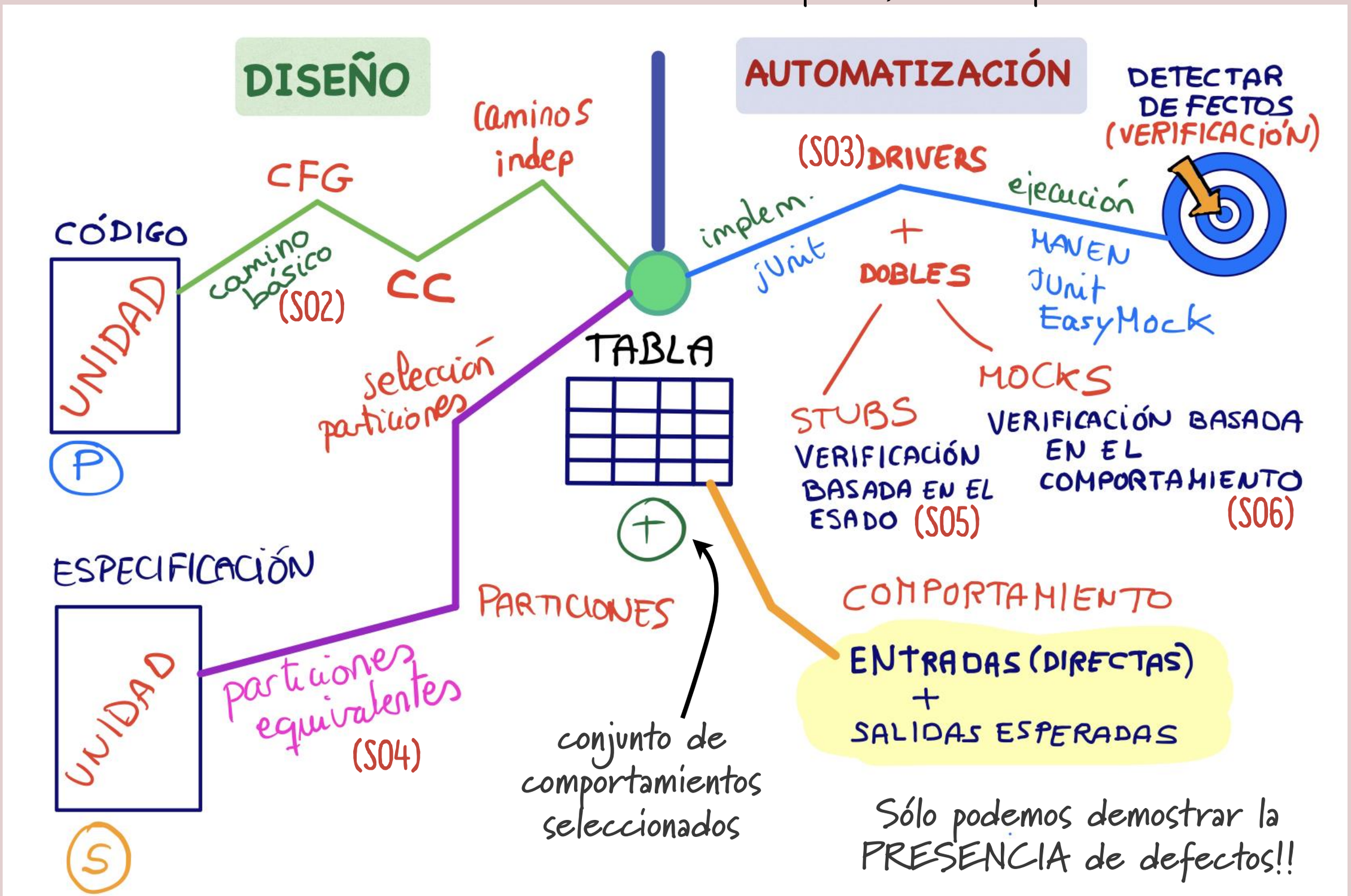
- Lenguaje JAVA
- Herram. construcción de proyectos: MAVEN
- Frameworks: JUnit, EasyMock

REPASO SESIONES S01..S06

El proceso de DISEÑO consiste en seleccionar, de forma sistemática, un conjunto de comportamientos a probar. El conjunto obtenido será efectivo y eficiente

No podemos hacer pruebas exhaustivas!!!

El proceso de AUTOMATIZACIÓN consiste en la ejecución de los comportamientos seleccionados, realización de las verificaciones correspondientes y obtención del informe de prueba, todo ello "pulsando un botón"



REPASO SESIONES P01..P06

Las sesiones prácticas nos permitirán comprender mejor los conceptos teóricos

SESIÓN P01

Durante el curso vamos realizar pruebas DINÁMICAS. Para ello necesitaremos AUTOMATIZAR la ejecución de casos de prueba. Necesitaremos una herramienta de CONSTRUCCIÓN DE PROYECTOS. Usaremos MAVEN.
Maven proporciona 3 build scripts denominados ciclos de vida, y que podemos configurar usando el fichero pom.xml, en el que tendremos que reconocer 4 "zonas": coordenadas, propiedades, dependencias y build.
Todos los proyectos maven tienen una estructura de directorios predefinida y fija.
El proceso de construcción termina con BUILD SUCCESS o BUILD FAILURE, y genera el directorio target
Los ficheros usados y/o generados por Maven se denominan artefactos y se identifican por sus coordenadas.

SESIÓN P02

Comportamiento = datos de entrada+resultado esperado

Podemos diseñar los casos de prueba a partir del código de nuestro SUT usando el método del camino básico. DISEÑO

Seleccionaremos un conjunto de comportamientos programados que garantizan la ejecución de todas las líneas de código (al menos una vez) de nuestro SUT y que se ejercitan todas las condiciones del SUT en su vertiente verdadera y falsa. PRUEBAS UNITARIAS

El conjunto obtenido es efectivo y eficiente

SESIÓN P03

AUTOMATIZACIÓN

Usaremos JUnit 5 para implementar drivers..
Un driver ejecuta un comportamiento seleccionado previamente (diseñado) Podremos parametrizarlos, reduciendo así la duplicación de código.
Disponemos de diferentes sentencias assert para verificar el resultado de los tests.
También podemos ejecutarlos de forma selectiva usando etiquetas. PRUEBAS UNITARIAS

JUnit genera un informe con 3 posibles resultados.
Compilaremos y ejecutaremos los tests a través de Maven incluyendo la librería JUnit en el pom.

REPASO SESIONES P01..P06

Queremos probar cada UNIDAD por SEPARADO!!!

SESIÓN P04

DISEÑO — PRUEBAS UNITARIAS

Podemos diseñar los casos de prueba a partir de la especificación de nuestro SUT (método de particiones equivalentes).
Seleccionaremos un conjunto de comportamientos especificados que garantizan la ejecución de todas las particiones de entrada/salida (al menos una vez), y que las particiones inválidas se prueban de una en una.
El conjunto obtenido es efectivo y eficiente

SESIÓN P05

AUTOMATIZACIÓN

Implementamos drivers usando VERIFICACIÓN basada en el ESTADO.
Usaremos dobles, que sustituirán a las dependencias externas de nuestro SUT durante las pruebas.
Los dobles (STUBS) nos permiten controlar las entradas indirectas a nuestro SUT, para así poder AISLAR la unidad a probar (método java).
Para poder inyectar los dobles durante las pruebas puede que necesitemos refactorizar nuestro SUT.
La implementación del doble está separada del código del driver

Maven se encargará de ejecutar las pruebas de forma automática!!!

PRUEBAS UNITARIAS

SESIÓN P06

AUTOMATIZACIÓN

Implementamos drivers usando VERIFICACIÓN basada en el COMPORTAMIENTO.
Usaremos dobles, que sustituirán a las dependencias externas de nuestro SUT durante las pruebas.
Los dobles (MOCKS) nos permiten verificar la interacción de nuestro SUT con las dependencias externas, AISLANDO el código de la unidad a probar (método java).
Para poder inyectar los dobles durante las pruebas puede que necesitemos refactorizar nuestro SUT.
Usaremos la librería EasyMock para implementar los dobles "dentro" del driver.
La librería EasyMock permite implementar los dos tipos de verificaciones.