

## Sesión S08: Pruebas del sistema y aceptación



Maria Isabel Alfonso Galipienso  
Universidad de Alicante [eli@ua.es](mailto:eli@ua.es)

### Pruebas del sistema

- Objetivo: encontrar defectos en las funcionalidades (desde el punto de vista del desarrollador)
- Diseño:
  - Método basado en casos de uso

### Pruebas de aceptación

- Objetivo: validar las expectativas del cliente
- Propiedades emergentes funcionales (siempre desde el punto de vista del usuario) y criterios de aceptación
- Diseño:
  - Método basado en escenarios

### Automatización de pruebas del sistema y de aceptación

- Aplicaciones con interfaz web: Selenium webdriver

Vamos al laboratorio...

P

## VALIDACIÓN

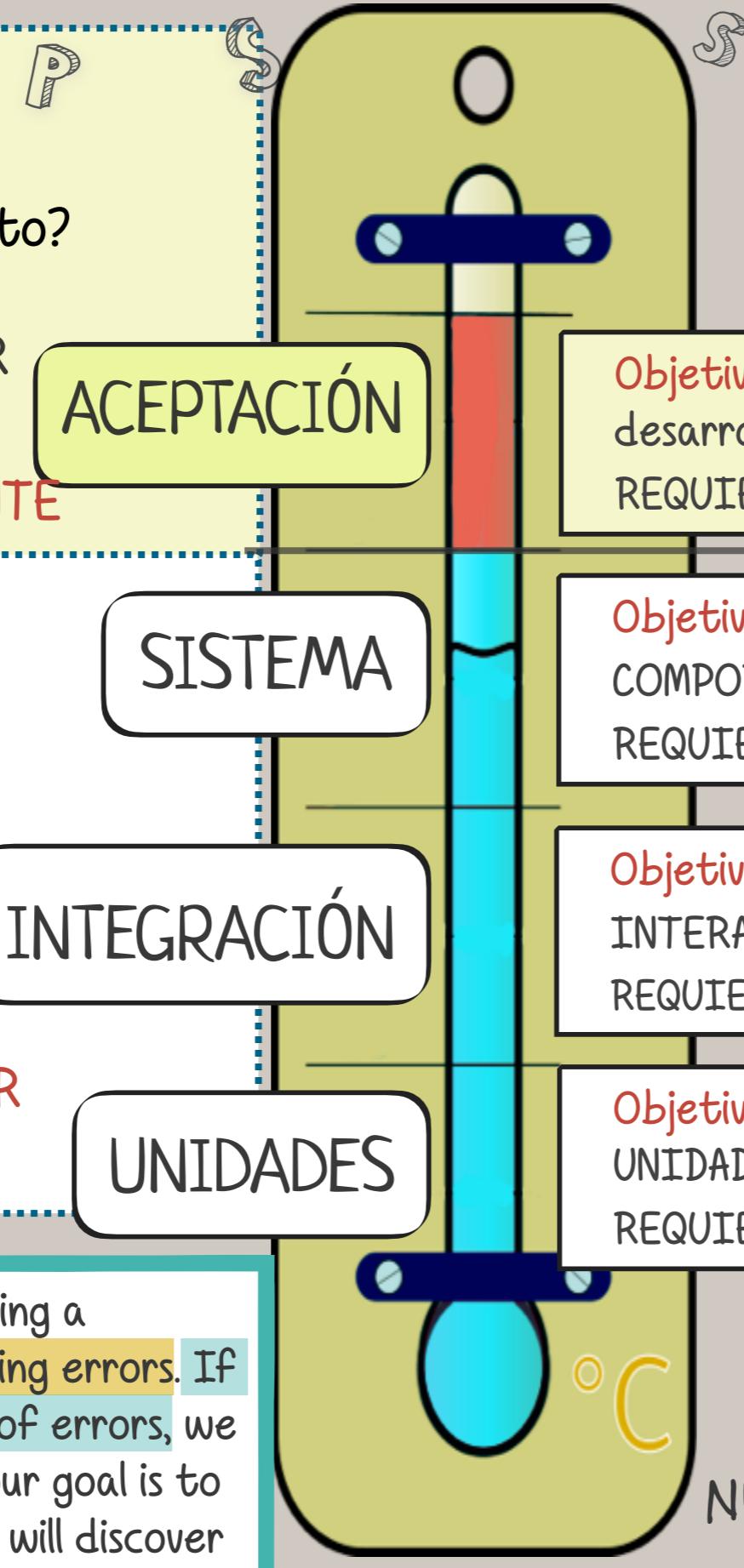
¿el producto es el correcto?

El objetivo es DEMOSTRAR que se satisfacen las EXPECTATIVAS DEL CLIENTE

## VERIFICACIÓN

¿el producto está implementado correctamente?

El objetivo general (intención) es ENCONTRAR DEFECTOS



## NIVELES DE PRUEBAS

Cada nivel tiene una granularidad y objetivos concretos diferentes. Hay un ORDEN temporal entre ellos.

**Objetivo:** valorar en qué grado el software desarrollado satisface las expectativas del cliente  
REQUIERE los CRITERIOS DE ACEPTACIÓN

**Objetivo:** encontrar DEFECTOS derivados del COMPORTAMIENTO del sw como un todo  
REQUIERE los REQUISITOS (funcionalidades) del sistema

**Objetivo:** encontrar DEFECTOS derivados de la INTERACCIÓN de las unidades probadas  
REQUIERE establecer un ORDEN de las unidades a integrar

**Objetivo:** encontrar DEFECTOS en el código de las UNIDADES probadas  
REQUIERE AISLAR el código de cada unidad a probar

"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)

NUESTRA intención es MUY IMPORTANTE!!!

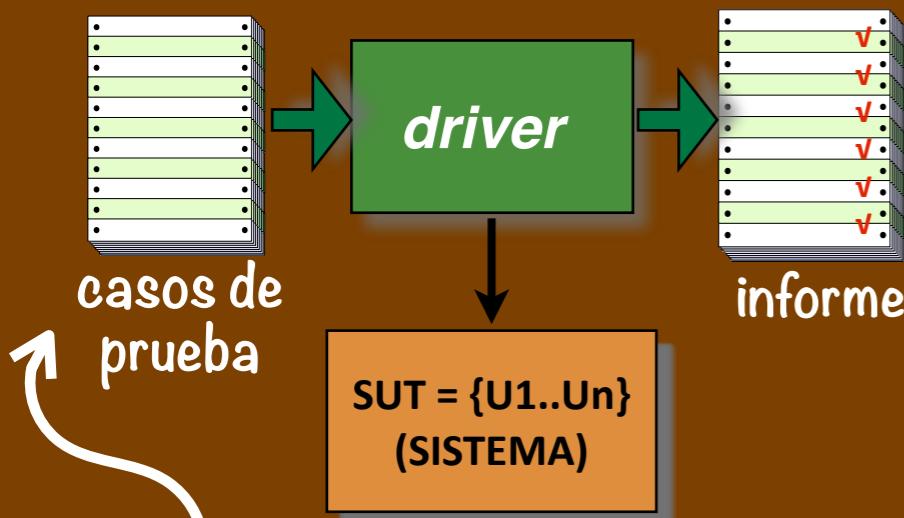


# PS PR. SISTEMA VS PR. ACEPTACIÓN SP

**VALIDACIÓN**

## VERIFICACIÓN

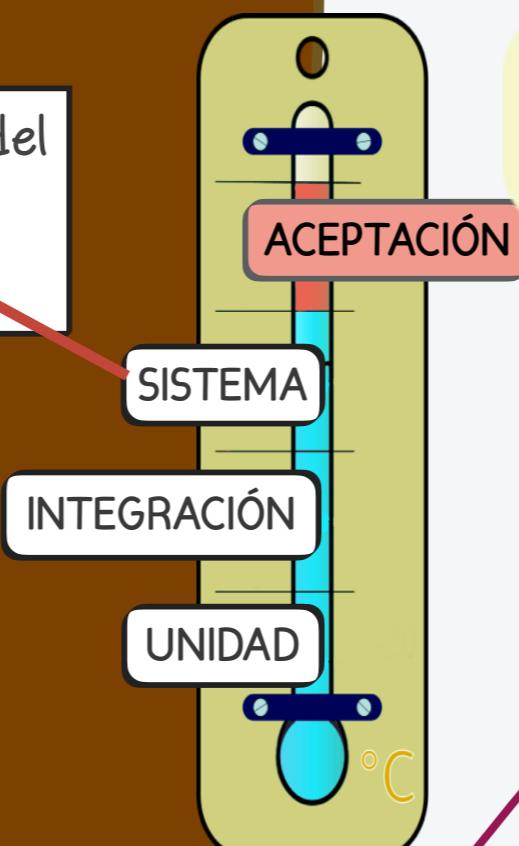
Objetivo: encontrar **DEFECTOS** derivados del **COMPORTAMIENTO** del sw como un todo  
REQUIERE los **REQUISITOS** del sistema



Los casos de prueba se diseñan desde el **punto de vista del DESARROLLADOR**

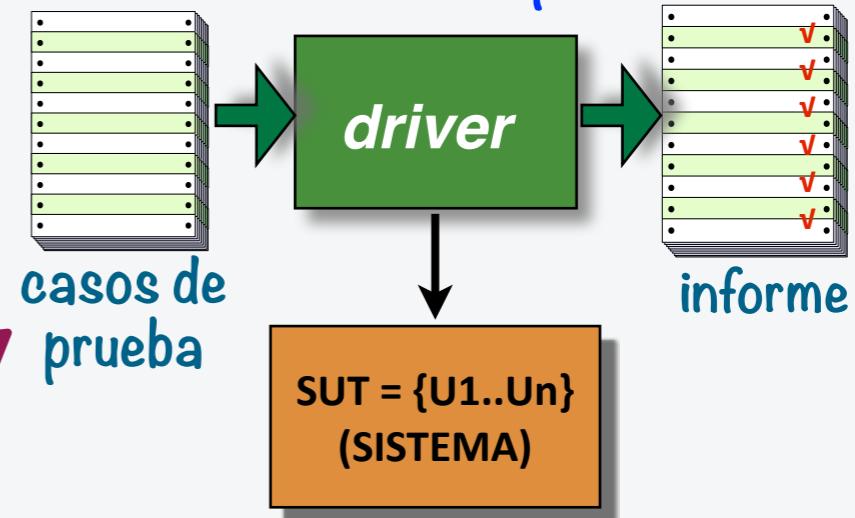


Los casos de prueba se diseñan desde el **punto de vista del USUARIO (CLIENTE)**

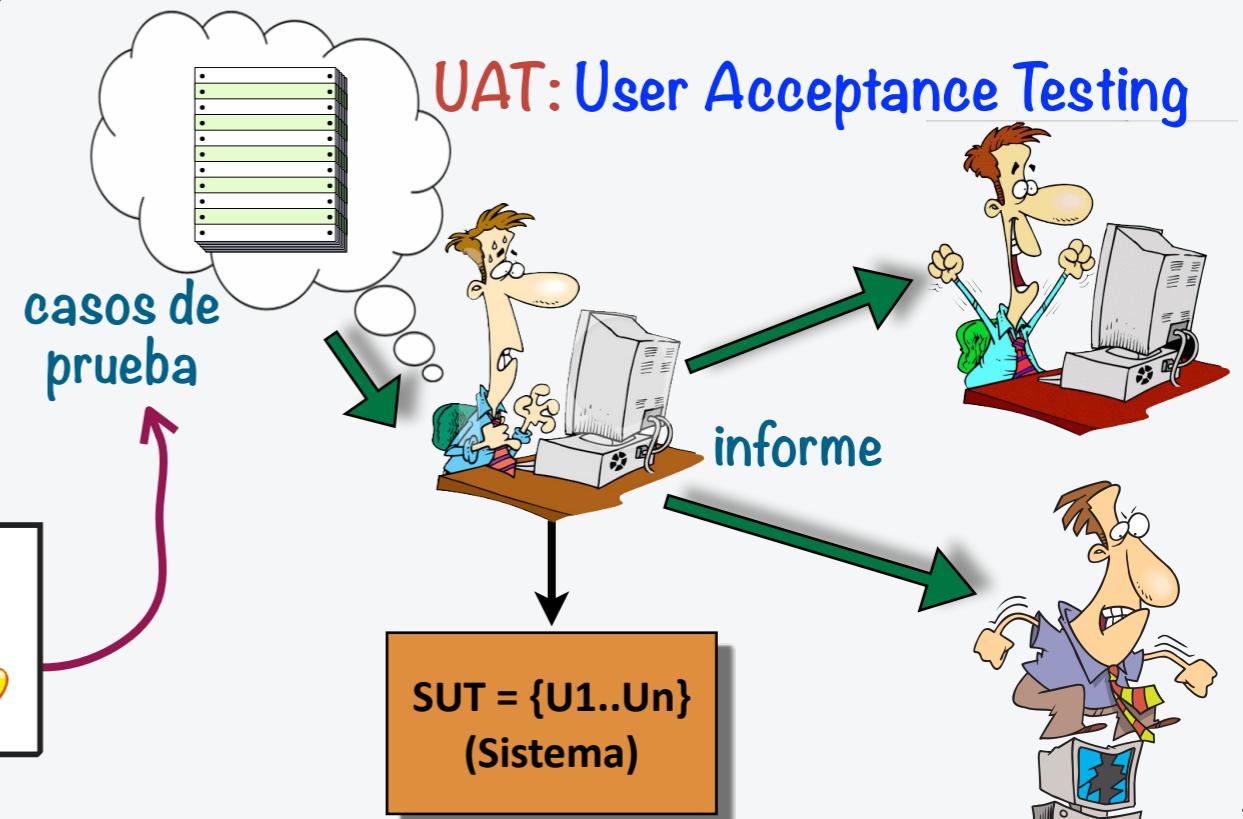


Objetivo: valorar en qué grado el software desarrollado satisface las expectativas del cliente  
REQUIERE los **CRITERIOS DE ACEPTACIÓN**

**BAT: Business Acceptance Testing**



**UAT: User Acceptance Testing**



# PRUEBAS DEL SISTEMA

P ver Cap. 8 Sommerville. 10ed "Software testing"

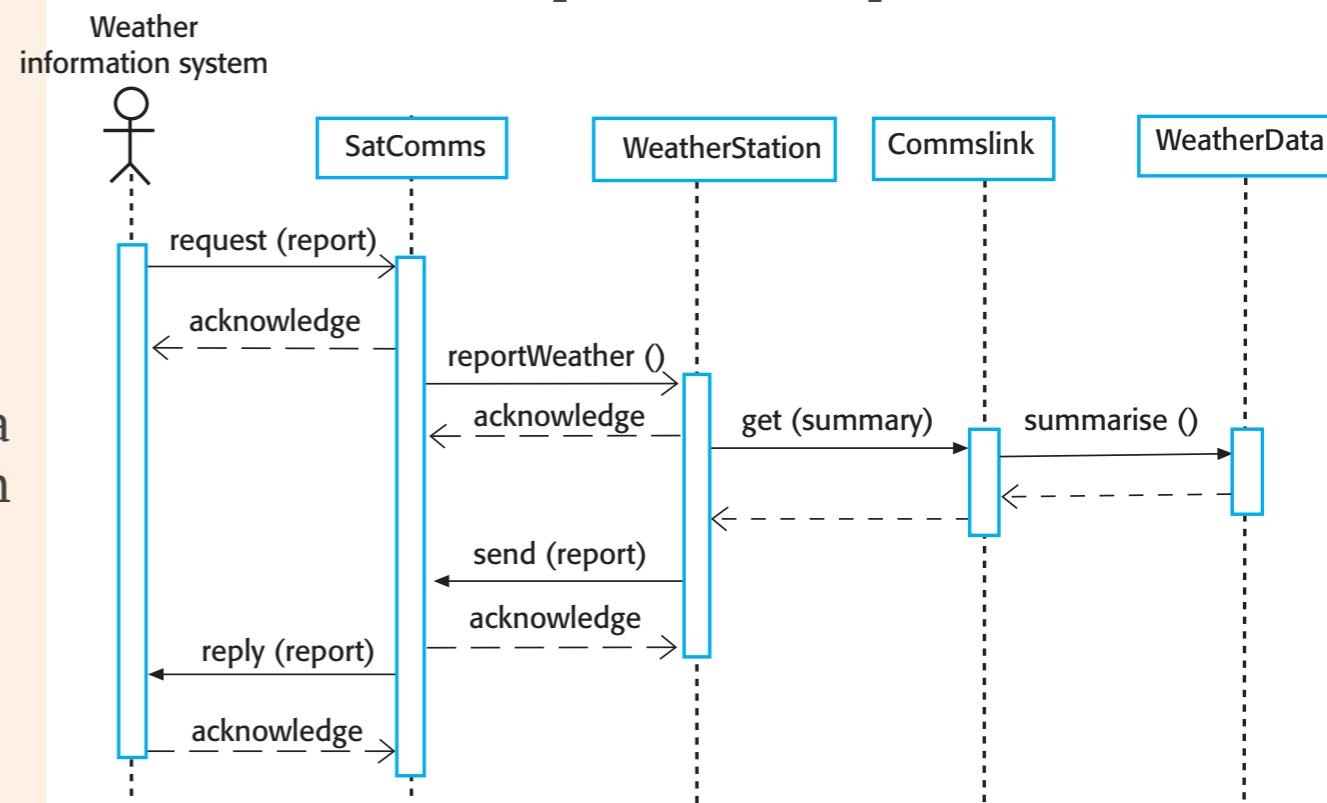
- Son pruebas dinámicas y obtienen los casos de prueba a partir de las especificaciones del sistema (caja negra)
- Se realizan durante el proceso de desarrollo, y usan todos los componentes del sistema, una vez que éstos han sido integrados. Se prueban:
  - Si los componentes son compatibles, si interactúan correctamente y transfieren el dato adecuado en el instante de tiempo adecuado a través de sus interfaces
- Las diferencias con las pruebas de integración son:
  - Se incluyen componentes reutilizables desarrollados por "terceros"
  - Los comportamientos a probar son los especificados para el sistema en su conjunto. P.ej. en un sistema de compra on-line, se prueba el proceso "completo" de compra por parte de un usuario
- Ejemplo de métodos de diseño del sistema:
  - Método de diseño basado en **casos de uso**
  - Método de diseño de **transición de estados**

**componente** = UNIDAD o conjunto de UNIDADES



Recuerda que "vemos" el sistema desde el punto de vista del DESARROLLADOR!!

- Diseño basado en **CASOS DE USO**:
  - Permiten probar las interacciones entre componentes
  - Cada caso de uso se implementa utilizando varios componentes del sistema. Dichas interacciones se muestran en los diagramas de secuencia asociados al caso de uso
  - La prueba de un caso de uso "fuerza" el que se lleven a cabo las diferentes interacciones entre los componentes implicados



# DISEÑO DE PRUEBAS DE SISTEMA

P

S

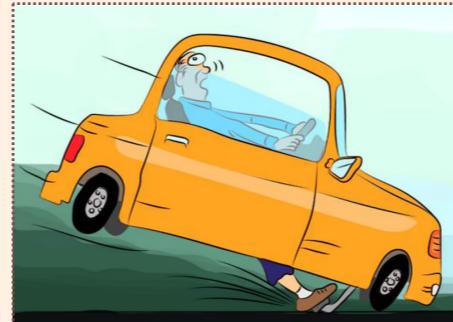
## Sobre el diseño de basado en **casos de uso**:

- Los diagramas de secuencia nos ayudan a diseñar los casos de prueba adecuados, ya que muestran qué entradas se requieren y qué salidas deben producirse entre los componentes
- Puesto que no podemos realizar pruebas exhaustivas, ¿Cuántos casos de prueba necesitamos diseñar para dar eficiencia a nuestro proceso de pruebas? En este caso, se deben fijar políticas de pruebas para elegir de forma adecuada un sub-conjunto de pruebas efectivo, como por ejemplo:
  - Todas las funciones del sistema que sean accedidas desde menús deben ser probadas
  - Combinaciones de funciones que sean accedidas a través del mismo menú, deben ser probadas
  - Siempre que se proporcionan entradas de usuario, se deben probar entradas correctas e incorrectas
  - Las funciones más utilizadas en un uso normal, deben ser las más probadas

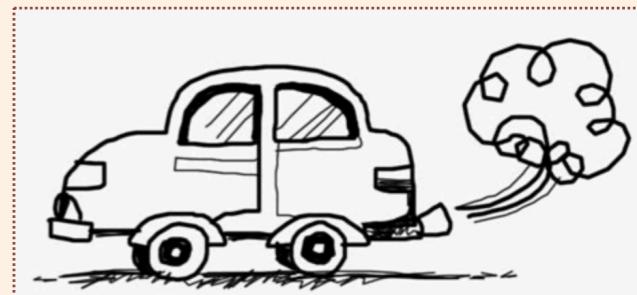
## Ejemplos de comportamientos ejercitados en pruebas de **SISTEMA**:



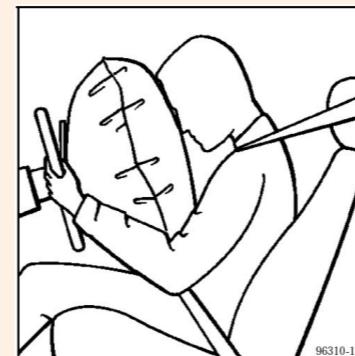
punto de vista del  
**DESARROLLADOR**



BRAKE TESTING



SMOKE TESTING

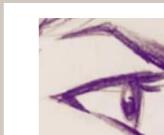


AIRBAG TESTING

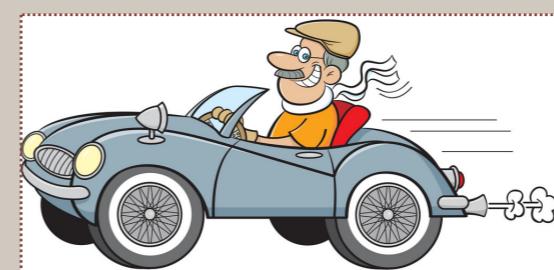


DIAGNOSE CAR TESTING

## Ejemplos de comportamientos ejercitados en pruebas de **ACEPTACIÓN**:



punto de vista del  
**USUARIO**



CONVERTIBLE TESTING



AUDIO CAR TESTING



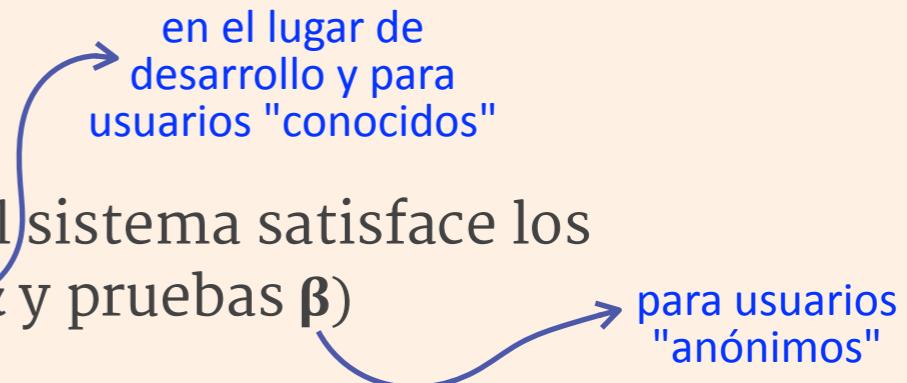
FAMILY CAR DRIVING  
TESTING

# ACCEPTANCE TESTING

El sistema se prueba desde el "punto de vista" del usuario

- Un producto está listo para ser entregado (delivered) al cliente después de que se hayan realizado las pruebas del sistema
  - A continuación, los clientes ejecutan los tests de **aceptación** basándose en sus expectativas sobre el producto. Finalmente, si los tests de aceptación son "aceptados" por el cliente, el sistema será puesto en PRODUCCIÓN.
- Las pruebas de aceptación son pruebas formales orientadas a determinar si el sistema satisface los criterios de aceptación (**acceptance criteria**)
  - Los **criterios de aceptación** de un sistema deben satisfacerse para ser aceptados por el cliente (éste generalmente se reserva el derecho de rechazar la entrega del producto si los tests de aceptación no "pasan")
- Hay dos categorías de pruebas de aceptación:
  - **User acceptance testing (UAT)**
    - \* Son **dirigidas por el cliente** para asegurar que **el sistema** satisface los criterios de aceptación contractuales (pruebas  $\alpha$  y pruebas  $\beta$ )
  - **Business acceptance testing (BAT)**
    - \* Son **dirigidas por la organización** que desarrolla el producto para asegurar que el sistema eventualmente "pasará" las UAT. Son un ensayo de las UAT en el lugar de desarrollo

Los criterios de aceptación se definen en etapas tempranas del desarrollo, pero los probamos al final del desarrollo, y después de haber verificado!!!



# ACCEPTANCE CRITERIA

Los criterios de aceptación se refieren a TODO el sistema, no a una parte del mismo

- Los criterios de aceptación deben ser DEFINIDOS y ACORDADOS entre el proveedor (organización a cargo del desarrollo) y el cliente
  - Constituyen el "núcleo" de cualquier acuerdo contractual entre el proveedor y el cliente
- Una cuestión clave es: ¿Qué criterios debe satisfacer el sistema para ser aceptado por el cliente?
  - El principio básico para diseñar los criterios de aceptación es asegurar que la calidad del sistema es aceptable
  - Los criterios de aceptación deben ser medibles, y por lo tanto, cuantificables
- Algunos **atributos de calidad** que pueden formar parte de los criterios de aceptación son:
  - **Corrección funcional y Completitud**
    - \* ¿El sistema hace lo que se quiere que haga? ¿Todas las características especificadas están presentes?
  - **Exactitud**, integridad de datos, **rendimiento**, **fiabilidad** y disponibilidad, mantenibilidad, robustez, confidencialidad, escalabilidad,...

# PROPIEDADES EMERGENTES

S

Sólo son visibles después de haber integrado TODO el sistema

P

- Cualquier atributo incluido en los criterios de aceptación es una **propiedad emergente**
- Las propiedades "emergentes" son aquellas que no pueden atribuirse a una parte específica del sistema, sino que "emergen" solamente cuando los componentes del sistema han sido integrados, ya que son el resultado de las complejas interacciones entre sus componentes. Por lo tanto, no pueden "calcularse" directamente a partir de las propiedades de sus componentes individuales
- Hay dos tipos de propiedades emergentes:
  - **Funcionales**: ponen de manifiesto el propósito del sistema después de integrar sus componentes
  - **No funcionales**: relacionadas con el comportamiento del sistema en su entorno habitual de producción (p.ej. fiabilidad, seguridad...)
- En esta sesión nos vamos a centrar en las pruebas de propiedades emergentes FUNCIONALES

Nos centraremos en las propiedades FUNCIONALES, pero recuerda que diseñaremos las pruebas sin tener en cuenta consideraciones técnicas o detalles internos de la aplicación. Consideraremos siempre el punto de vista del usuario.

# DISEÑO DE PRUEBAS DE ACEPTACIÓN

Los casos de prueba requieren (como siempre) datos de entrada y resultados esperados CONCRETOS

- Deberían ser responsabilidad de un grupo separado de pruebas que no esté implicado en el proceso de desarrollo. Se trata de determinar que el sistema es lo suficientemente "bueno" como para ser usado (entregado al cliente, o ser lanzado como producto): cumple los criterios de aceptación
- Normalmente se trata de un proceso black-box (functional testing) basado en la especificación del sistema. También reciben el nombre de "pruebas funcionales", para indicar que se centran en la "funcionalidad" y no en la implementación
- Ejemplo de **método de diseño** de pruebas emergentes funcionales:
  - Diseño de pruebas basado en **requerimientos**
    - \* son pruebas de validación (se trata de demostrar que el sistema ha implementado de forma adecuada los requerimientos y que está preparado para ser usado por el usuario). Cada requerimiento debe ser "testable"
  - Diseño de pruebas de **escenarios**
    - \* los escenarios describen la forma en la que el sistema debería usarse
    - \* un escenario incluye (contempla) varios requerimientos

Los DATOS de entrada pueden incluir "secuencias" de acciones llevadas a cabo por el usuario. Nuestros drivers tendrán muchas más líneas de código!!

# DISEÑO DE PRUEBAS BASADO EN REQUERIMIENTOS (I)

Capítulo 8.3.1 "Software Engineering" 9th. Ian Sommerville

- Un principio general de una buena especificación de un requerimiento es que debe escribirse de forma que se pueda diseñar una prueba a partir de él
  - [610-12-1990-IEEE Standard Glossary of Software Engineering Terminology]. A requirement is:
    1. A condition or capability needed by a user to solve a problem or achieve an objective
    2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.
    3. A documented representation of a condition or capability as in 1 or 2.
- Ejemplo de requerimiento “testable”

*If a patient is known to be allergic to any particular medication, then prescription of that medication shall result in a warning message being issued to the system user.*

*If a prescriber chooses to ignore an allergy warning, they shall provide a reason why this has been ignored.*

- Ejemplo de casos de prueba que podemos derivar del requerimiento anterior:
  1. Set up a patient record with no known allergies. Prescribe medication for allergies that are known to exist. Check that a warning message is not issued by the system

# DISEÑO DE PRUEBAS BASADO EN REQUERIMIENTOS (II)

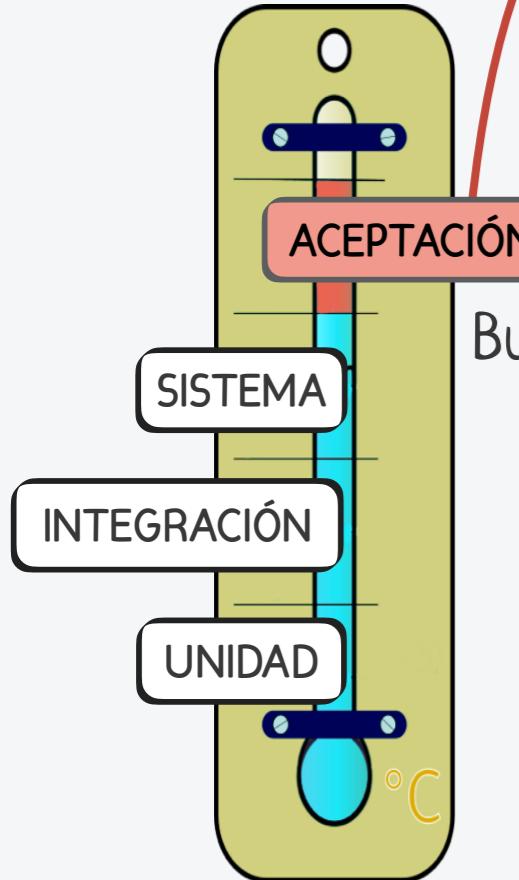
- Ejemplo de casos de prueba que podemos derivar del requerimiento anterior (continuación):

2. Set up a patient record with a known allergy. Prescribe the medication to that the patient is allergic to, and check that the warning is issued by the system.
3. Set up a patient record in which allergies to two or more drugs are recorded. Prescribe both of these drugs separately and check that the correct warning for each drug is issued.
4. Prescribe two drugs that the patient is allergic to. Check that two warnings are correctly issued.
5. Prescribe a drug that issues a warning and overrule that warning. Check that the system requires the user to provide information explaining why the warning was overruled.

- Fijaos que para un ÚNICO requerimiento necesitaremos VARIOS tests para asegurar que "cubrimos" todo el requerimiento

# PS PRUEBAS DE ACEPTACIÓN

**VALIDACIÓN** ¿el producto es el correcto?



Las pruebas de aceptación determinan en qué grado el sistema satisface los **CRITERIOS DE ACEPTACIÓN**

Business acceptance testing

User acceptance testing ( $\alpha, \beta$ )

Propiedades emergentes

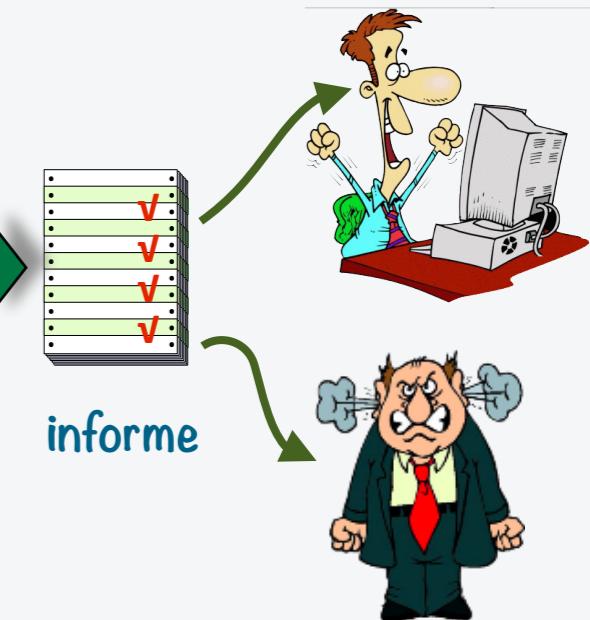
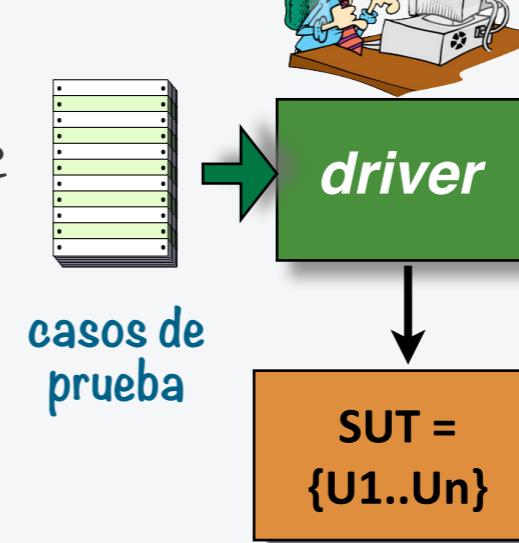
NO funcionales

Automatización de pruebas de aplicaciones Web

También podemos usar Selenium Webdriver para automatizar tanto las pruebas del SISTEMA como de ACEPTACIÓN

Don't  
FORGET!

Selenium WebDriver



Los casos de prueba en este nivel SIEMPRE se diseñan desde al punto del vista del usuario. Para automatizar estas pruebas, necesitamos herramientas que nos permitan reproducir las acciones del usuario sobre la aplicación.

El objetivo del tester en este nivel ya no es encontrar defectos, sino EVALUAR LA CALIDAD DEL SOFTWARE (recuerda los objetivos que atribuimos a un tester en el primer tema de la asignatura)



# CARACTERÍSTICAS DE SELENIUM WEBDRIVER



<https://www.selenium.dev/documentation/webdriver/>

- Proporciona un buen control del navegador a través de implementaciones específicas para cada uno de ellos
- Permite realizar una programación de los tests usando diferentes lenguajes

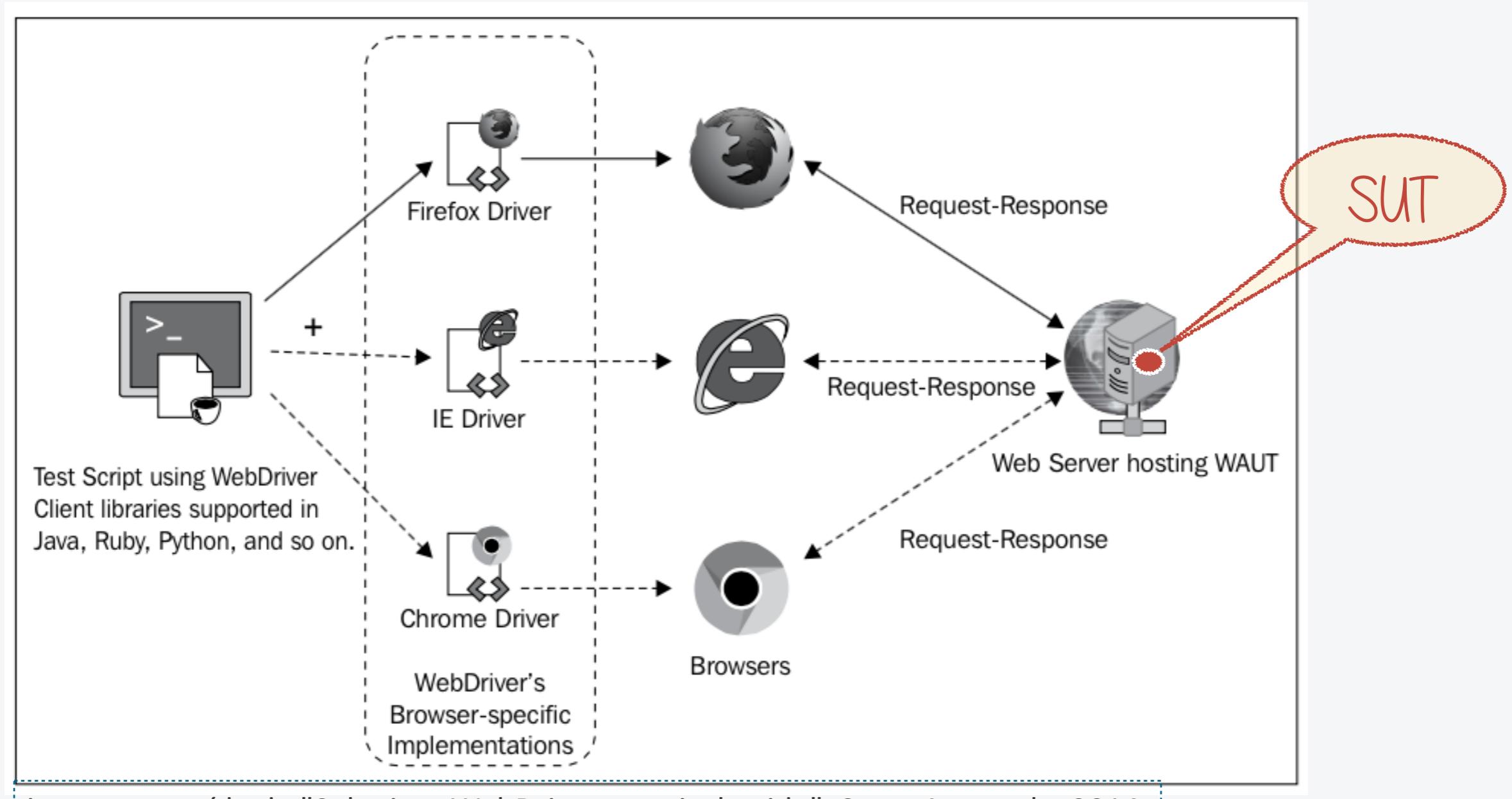


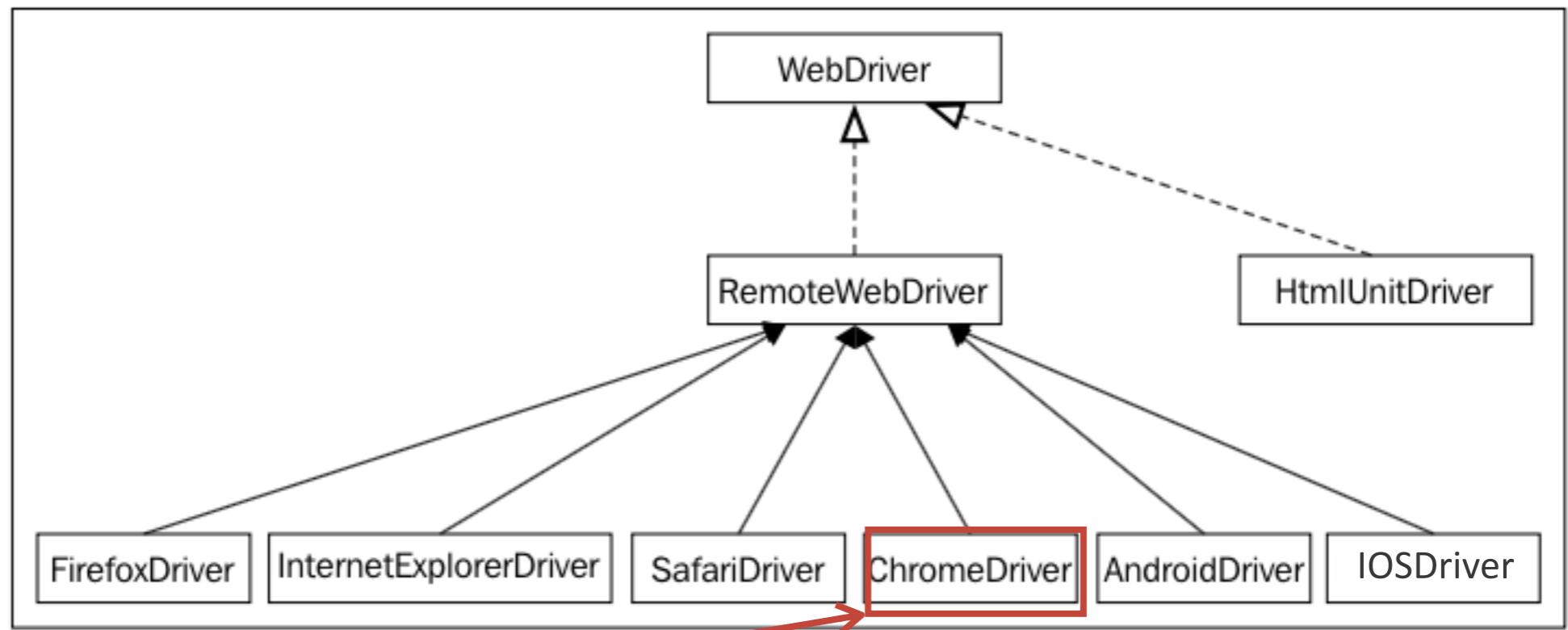
Imagen extraída de "Selenium WebDriver practical guide". Satya Avasarala. 2014.

WAUT: Web Application Under Test

# WEBDRIVER INTERFACE Y WEBELEMENTS

<https://www.selenium.dev/selenium/docs/api/java/overview-summary.html>

- WebDriver es una interfaz cuya implementación concreta la realizan dos clases: RemoteWebDriver y HtmlUnitDriver



Sesión 8: Pruebas sistemáticas+aceptación

```
WebDriver driver = new ChromeDriver(); → "driver" representa el navegador (se abre una "sesión")
driver.get("http://www.google.com"); → realizamos una acción sobre el navegador
WebElement searchBox = driver.findElement(By.name("q")); → localizamos un elemento en la página
searchBox.sendKeys("Packt Publishing");
searchBox.submit();
driver.quit(); → cerramos la sesión, también se cierra el navegador
```

interaccionamos con los elementos de la página

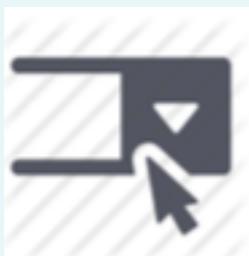
- Una página web está formada por elementos HTML, que son objetos de tipo **WebElement** en el contexto de WebDriver
- Una vez localizados los WebElements, podremos realizar **acciones** sobre ellos

# ELEMENTOS HTML MÁS USADOS



<https://www.w3schools.com/html/>

Los objetos WebElement representan elementos HTML



<select> <option ...



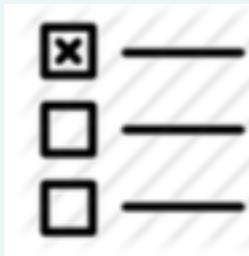
Drop Down List



<input type="radio" ...



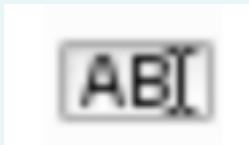
Radio Button



<input type="checkbox" ...



Check Box



<input type="text" ...



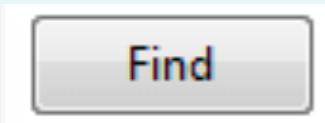
Text Box



<input type="password" ...



Password Box



<input type="button" ...



Button

# MECANISMOS DE LOCALIZACIÓN

<https://www.selenium.dev/.../elements/finders/>

<https://www.selenium.dev/.../locators/>

- Antes de realizar alguna acción (entradas del usuario) con WebDriver, debemos LOCALIZAR el elemento que nos interese. Para ello utilizamos los métodos:

`WebElement findElement(By by) throws NoSuchElementException` ← unchecked exception

- `findElement()` devuelve el primer elemento encontrado (es posible que haya varios elementos que cumplan el criterio de localización usado)

`java.util.List<WebElement> findElements(By by)`

- `findElements()` devuelve todos los elementos encontrados (o una lista vacía)
- En ambos casos como parámetro de entrada se requiere una instancia de un objeto de tipo `By`, que nos permite localizar elementos en una página web
- El **inspector** de elementos de Chrome puede ayudarnos a localizar los elementos HTML de las páginas web cargadas por el navegador. Es tarea del desarrollador elegir el "locator" adecuado para utilizarlo en `findElement/s()`

- Hay 8 formas de **localizar** un WebElement en una página web:

- `By.name()`, `By.id()`, `By.tagName()`, `By.className()`, `By.linkText()`,  
`By.partialLinkText()`, `By.xpath()`, `By.cssSelector()`
- Preferiblemente se debe usar `By.id()` siempre que sea posible, ya que el atributo id tiene un valor único.
- Si no podemos localizar un elemento por su id, la segunda mejor opción es usar **selectores css**.

# EJEMPLOS DE LOCALIZACIÓN

```

<html>
<body>
<style>
.information {
    background-color: white;
    color: black;
    padding: 10px;
}
</style>
<h2>Contact Selenium</h2>

<form action="/action_page.php">
    <input type="radio" name="gender" value="m" />Male &nbsp;
    <input type="radio" name="gender" value="f" />Female <br>
    <br>
    <label for="fname">First name:</label><br>
    <input class="information" type="text" id="fname" name="fname" value="Jane"><br><br>
    <label for="lname">Last name:</label><br>
    <input class="information" type="text" id="lname" name="lname" value="Doe"><br><br>
    <label for="newsletter">Newsletter:</label>
    <input type="checkbox" name="newsletter" value="1" /><br><br>
    <input type="submit" value="Submit">
</form>

<p>To know more about Selenium, visit the official page
<a href ="www.selenium.dev">Selenium Official Page</a>
</p>

</body>
</html>

```

```

WebElement e1 = driver.findElement(By.id("lname"));
WebElement e2 = driver.findElement(By.name("newsletter"));
WebElement e3 = driver.findElement(By.className("information"));
WebElement e4 = driver.findElement(By.tagName("a"));
WebElement e5 = driver.findElement(By.xpath("/html/form/input[1]")); ← Path absoluto
WebElement e6 = driver.findElement(By.xpath("//input[@value='f']")); ← Path relativo
WebElement e7 = driver.findElement(By.partialLinkText("Official Page"));

```

## Contact Selenium

The screenshot shows a contact form with the following elements and their identifiers:

- Male radio button: e5
- Female radio button: e6
- First name input field: e3
- Last name input field: e1
- Newsletter checkbox: e2
- Submit button (not explicitly labeled with an identifier)
- Link to Selenium Official Page: e7



Es imprescindible localizar un elemento antes de interactuar con él

# SELECTORES CSS

[https://www.w3schools.com/cssref/css\\_selectors.php](https://www.w3schools.com/cssref/css_selectors.php)

- Los "css selectors" son patrones de caracteres utilizados para identificar un elemento HTML basado en una combinación de etiquetas HTML, id, class, y otros atributos. Los formatos más comunes para los selectores css son:

- \* #id → "#" seguida del valor del atributo id
- \* tag.class → valor de etiqueta seguida ":" y del valor del atributo class
- \* tag.class[attribute=value]
- \* tag[attribute=value]
- \* [attribute\*=value] → "\*=" indica que el valor del atributo contiene el valor especificado
- \* .class1.class2 → elementos cuyo atributo class tiene los valores class1 y class2
- \* tag1 tag2 → etiqueta con valor tag2 que sea descendiente de tag1
- \* tag1 > tag2 → etiqueta con valor tag2 que sea hija de tag1

```
1 <html>
2   <body>
3     <form id="loginForm">
4       <input class="required" name="username" type="text" />
5       <input class="required passfield" name="password" type="password" />
6       <input name="continue" type="submit" value="Login" />
7       <input name="continue" type="button" value="Clear" />
8     </form>
9   </body>
10  </html>
```

- form#loginForm (línea 3)
- input[name="username"] (4)
- .required.passfield (5)
- input.required[type="text"] (4)
- input.passfield (5)

```
<div class="form-group">
  <button id="login-button" type="submit" class="btn btn-dark submit-btn"
    data-callback="onSubmit">Login</button>
</div>
```

- button.submit-btn[data-callback="onSubmit"]

```
<input type="email" name="email" value="" placeholder="Email" required="required"
  autofocus="autofocus" class="form-control mt-3 form-control-lg">
```

- input[class\*='control']

# P ACCIONES SOBRE LOS WEBELEMENTS

P Una vez que hemos localizado el elemento que nos interesa, podemos ejecutar ACCIONES sobre ellos.

- Cada tipo de elemento tiene asociado un conjunto diferente de posibles acciones. P.ej. sobre un elemento textbox, podemos introducir un texto o borrarlo

O Ejemplos de acciones:

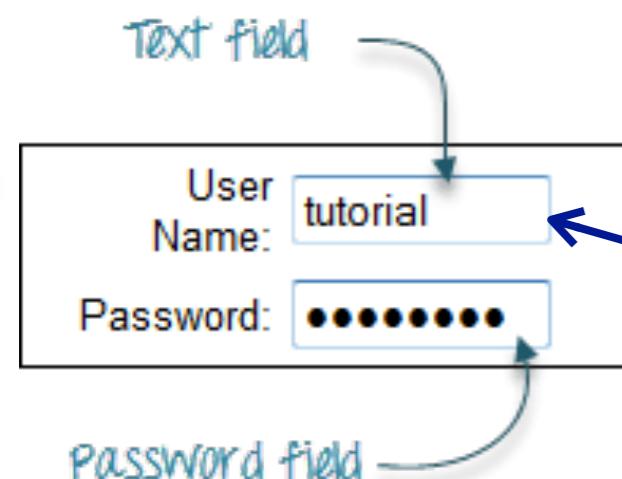
- **sendKeys**(secuencia de caracteres)
  - \* Se utiliza para introducir texto en elementos textbox o textarea
- **clear()** se utiliza para borrar texto en elementos textbox o textarea
- **submit()**
  - \* Puede aplicarse sobre un elemento form, o sobre un elemento que esté dentro de un form. Envía el formulario de la página web al servidor en el que reside la aplicación web

O Ejemplos de acciones que pueden ejecutarse sobre **cualquier** WebElement:

- `getAttribute()`, `getLocation()`, `getText()`, `isDisplayed()`, `isEnabled()`, `isSelected()`

# EJEMPLOS DE ACCIONES

<http://www.guru99.com/accessing-forms-in-webdriver.html>



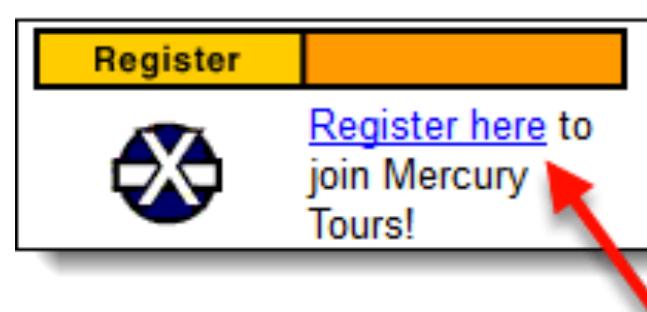
## Introducir texto en un text box y password box

```
driver.findElement(By.name("username")).sendKeys("tutorial");
```

combinación de etiquetas html, id, class y otros atributos

## seleccionar un radio box

```
driver.findElement(By.cssSelector("input[value='Business']")).click();
```



## pulsar sobre un enlace de texto

```
driver.findElement(By.linkText("Register here")).click();
```

```
driver.findElement(By.partialLinkText("here")).click();
```

Mercury Tours Registration Page

Country: UNITED STATES

## seleccionar elementos en un drop box

```
<td>
  <select size="1" name="country">
</td>
```

```
import org.openqa.selenium.support.ui.Select;
Select drpCountry =
  new Select(driver.findElement(By.name("country")));
drpCountry.selectByVisibleText("ANTARTICA");
```

# TIEMPOS DE ESPERA



No se deben "mezclar" ambos tipos de esperas!!

<https://www.selenium.dev/documentation/webdriver/waits/>

- P O Podemos establecer tiempos de espera en nuestros tests para evitar errores debidos a que no se localiza un elemento en la página porque todavía se esté cargando (excepción NoSuchElementException)
- P □ Tiempo de espera **implícito**: es común a todos los WebElements y tiene asociado un timeout global para todas las operaciones del driver
- Tiempo de espera **explicito**: se establece de forma individual para cada WebElement

```
...
WebDriver driver = new ChromeDriver();
driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
driver.get("www.google.com");
...
```

**timeout implícito**

```
...
//Create Wait using WebDriverWait.
//This will wait for 10 seconds for timeout before
//title is updated with search term
WebDriverWait wait = new WebDriverWait(driver,
                                         Duration.ofSeconds(10));
wait.until(ExpectedConditions.titleContains("selenium"));
...
```

**timeout explícito**

Si el elemento se carga antes del límite especificado se cancela el timeout

# MÚLTIPLES ACCIONES (GRUPOS DE ACCIONES)

Clase Actions

interfaz Action

Las acciones agrupadas se ejecutan de forma secuencial

- P
- Podemos indicar a WebDriver que realice múltiples acciones agrupándolas en una acción compuesta, siguiendo estos tres pasos:
    - Invocar la clase **Actions** para agrupar las acciones **(1)**
      - \* La clase Actions se utiliza para emular eventos complejos de usuario
    - Construir la acción (**Action**) compuesta por el conjunto de acciones **(2)** anteriores
    - Realizar (ejecutar) la acción compuesta **(3)**

```
WebDriver driver = new ChromeDriver();
driver.get("http://www.example.com");
WebElement one = driver.findElement(By.name("one"));
WebElement three = driver.findElement(By.name("three"));
WebElement five = driver.findElement(By.name("five"));
// Add all the actions into the Actions builder
Actions builder = new Actions(driver); (1)
// Generate the composite action
Action compositeAction = builder.keyDown(Keys.CONTROL)
    .click(one)
    .click(three)
    .click(five)
    .keyUp(Keys.CONTROL)
    .build(); (2)
// Perform the composite action.
compositeAction.perform(); (3)
```

En este ejemplo el usuario selecciona los tres elementos (manteniendo pulsada la tecla Ctrl mientras realiza la selección)

# EJEMPLOS DE ACCIONES BASADAS EN EL RATÓN

S

P

P

- El método **click()** se utiliza para simular que pulsamos el botón izquierdo del ratón:
  - **public Actions click()**
    - \* Pulsación del botón izquierdo del ratón, independientemente o no de que estemos sobre algún elemento de la página
    - \* Este método suele usarse combinado con otros, para crear una acción compuesta. P.ej.

```
WebElement one = driver.findElement(By.name("one")); Actions builder = new Actions(driver);
//Click on One
builder.moveToElement(one).click().build().perform();
```

- **public Actions click(WebElement onElement)**
  - \* Pulsación del botón izquierdo del ratón sobre un WebElement

```
WebElement one = driver.findElement(By.name("one")); Actions builder = new Actions(driver);
//Click on One
builder.click(one); builder.build().perform();
```

- Método **public Actions moveToElement( WebElement toElement)**

```
WebElement one = driver.findElement(By.name("one")); Actions builder = new Actions(driver);
//Click on One
builder.moveToElement(one).click().build().perform();
```

- Otros métodos que podemos utilizar son:

- \* **public Actions doubleClick();** (doble click con botón izquierdo)
- \* **public Actions contextClick();** (botón derecho del ratón)

# EJEMPLOS DE ACCIONES BASADAS EN EL TECLADO

P

P

- métodos `keyDown()` y `keyUp()`

- `public Actions keyDown(Keys theKey) throws IllegalArgumentException`

- \* Se genera una excepción si el argumento no es una de las teclas Shift, Ctrl, Alt

- `public Actions keyUp(Keys theKey)`

- método `sendKeys()`

- `public Actions sendKeys(CharSequence keysToSend)`

- \* Se utiliza para teclear caracteres en elementos de la página como text boxes, ...

- también se puede utilizar el método `WebElement.sendKeys(CharSequence k)`

- Ejemplo:

```
driver = new ChromeDriver();
driver.get(baseUrl);
WebElement linkText = driver.findElement(By.linkText("Element"));

Actions builder = new Actions(driver);
builder.contextClick(linkText) //activamos el menú contextual de linkText
    .sendKeys(Keys.ARROW_DOWN)
    .sendKeys(Keys.ENTER) //seleccionamos la primera de las opciones del menú
    .build()
    .perform();
```

# OPERACIONES DE NAVEGACIÓN

- Navegar a la página anterior: `driver.navigate().back()`
- Navegar a la página siguiente: `driver.navigate().forward()`
- Métodos de refresco: `driver.navigate().refresh()`
- Manejo de frames: `driver.switchTo.frame(index)`
- Manejo de ventanas: `driver.switchTo.window(window)`

## □ Ejemplo:

- \* Si tenemos varias ventanas:

```
driver.get(baseUrl);
String window1 = driver.getWindowHandle();
System.out.println("First Window Handle is: "+window1);

link = driver.findElement(By.linkText("Google Search"));
link.click();
String window2 = driver.getWindowHandle();
System.out.println("Second Window Handle is: "+window2);
System.out.println("Number of Window Handles so far: "
    +driver getWindowHandles().size());
driver.switchTo().window(window1);
```

- \* Si tenemos varias pestañas en una única ventana:

```
//Open a new tab using Ctrl + t
driver.findElement(By.cssSelector("body")).sendKeys(Keys.CONTROL +"t");
//Switch between tabs using Ctrl + \t
driver.findElement(By.cssSelector("body")).sendKeys(Keys.CONTROL +"\t");
```

@Test

```
public void accesoUAC1() {  
    WebDriver driver = new ChromeDriver();  
    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));  
  
    driver.get("https://www.ua.es");  
    //En la página hay dos elementos con el mismo enlace  
    List<WebElement> enlacesEstudios=driver.findElements(By.linkText("Estudios"));  
    //queremos acceder al segundo de ellos  
    enlacesEstudios.get(1).click();  
  
    WebElement enlaceGrados= driver.findElement(By.linkText("Grados Oficiales"));  
    enlaceGrados.click();  
  
    Assertions.assertTrue(driver.getTitle().contains("Grados Oficiales"));  
    WebElement buscadorAsignaturas= driver.findElement(By.linkText("BUSCADOR DE ASIGNATURAS"));  
    JavascriptExecutor jse = (JavascriptExecutor) driver;  
    //Below code will scroll the page till the element is found  
    jse.executeScript("arguments[0].scrollIntoView()", buscadorAsignaturas);  
    //ahora ya tenemos el elemento visible y podemos hacer click sobre el  
    buscadorAsignaturas.click();  
  
    Assertions.assertTrue(driver.getTitle().contains("Buscador de Asignaturas"));  
    WebElement campoCodigo = wait.until(presenceOfElementLocated(By.id("TextCodAsi")));  
    campoCodigo.sendKeys("34027");  
    WebElement boton=driver.findElement(By.id("ButBuscar"));  
    //hacemos scroll hasta ver el botón  
    jse.executeScript("arguments[0].scrollIntoView()", boton);  
    //ahora ya tenemos el elemento visible y podemos hacer click sobre el  
    boton.click();  
  
    WebElement enlacePpss= driver.findElement(By.partialLinkText("PLANIFICACIÓN Y PRUEBAS"));  
    Assertions.assertEquals("PLANIFICACIÓN Y PRUEBAS DE SISTEMAS SOFTWARE", enlacePpss.getText());  
    driver.quit();  
}
```

# WEBDRIVER Y MAVEN

P

S

S

P

- Necesitamos incluir la dependencia con la librería de WebDriver en nuestro proyecto Maven:

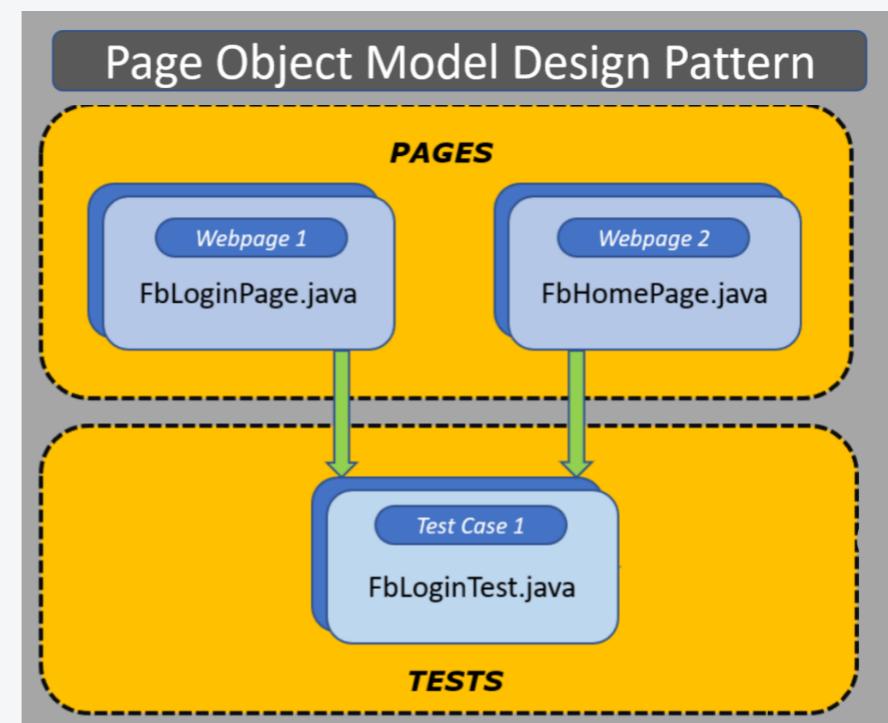
```
<dependencies>
  <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>4.31.0</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

- ¿Dónde implementaremos nuestros tests de aceptación?
  - **Opción 1:** en `src/test/java`, junto con el resto de drivers del proyecto. Serán tests ejecutados por `failsafe`. En `src/main/java` tendremos el código fuente de nuestro proyecto
  - **Opción 2:** en `src/test/java` de un proyecto maven independiente (el proyecto únicamente contiene los tests de aceptación). Serán tests ejecutados por `surefire`. En `src/main/java` tendremos código que usarán nuestros tests (clases del patrón de diseño **page object**). Esta opción la usaremos sólo si NO tenemos en `src/main` los fuentes de nuestro proyecto.

# MANTENIBILIDAD DE NUESTROS TESTS

[https://www.selenium.dev/documentation/test\\_practices/encouraged/page\\_object\\_models/](https://www.selenium.dev/documentation/test_practices/encouraged/page_object_models/)

- Los tests implementados para nuestra aplicación web, funcionarán siempre y cuando no se produzcan cambios en la vista web (código de las páginas web)
  - Si una o más páginas de nuestra aplicación web sufren cambios, tendremos que cambiar el código de nuestros tests (probablemente en muchos de ellos). P.ej. supongamos que un elemento de la página cambia su ID. Si dicho elemento es accedido desde N tests, tendremos que refactorizar todos ellos
- Para facilitar la mantenibilidad, y reducir la duplicación de código de nuestros tests es útil el patrón de diseño "Page Object Model" (POM)
  - La idea es independizar los tests de las páginas html
  - Básicamente consiste en crear una clase para cada página web, en la que:
    - \* sus miembros (atributos) serán los elementos de la página web correspondiente, y
    - \* sus métodos serán todos los SERVICIOS que nos proporciona la página



- El API de Webdriver proporciona varios elementos para implementar este patrón:
  - Anotación **@FindBy** para injectar los objetos que representan los elementos html de una página web
  - Clase **PageFactory** para obtener los objetos que representan las páginas html

# PAGE OBJECT PATTERN. EJEMPLO

ver <http://www.guru99.com/page-object-model-pom-page-factory-in-selenium-ultimate-guide.html>

- Supongamos que queremos implementar un test para una aplicación bancaria (<http://guru99.com/V4>)

## Guru99 Bank

UserID  
Password  
LOGIN    RESET

Servicio de login

Servicio de reset

- El test consistirá en:
- A. Accedemos a la url inicial de la aplicación
- B. Verificamos que estamos en la página correcta
- C. Nos logueamos con éxito
- D. Verificamos que accedemos a la página correcta

## Steps To Generate Access

- Visit - [here](#)
- Enter your email id
- Login credentials is allocated to you and mailed at your id
- Login credentials are only valid for 20 days! So Hurry Up and quickly complete your tasks

Servicio de info de acceso a la aplicación



- Los servicios se implementan como métodos
- Los elementos html se implementan como atributos del objeto PageObject

## LoginPage

userID

password

login

reset

info

elementos  
html

login()

reset()

servicios

inforegister()

# PÁGINA WEB CON LOS SERVICIOS DEL BANCO

P

P

**Guru99 Bank**

Manager

New Customer

Edit Customer

Delete Customer

New Account

Edit Account

Delete Account

Deposit

Withdrawal

Fund Transfer

Change Password

Balance Enquiry

Mini Statement

Customised Statement

Log out

Welcome To Manager's Page of Guru99 Bank  
Manger Id : mngr11212

La clase ManagerPage ahora depende del código html. Si cambia el código html sólo tenemos que hacer cambios en la PO, no en nuestro test

Llamaremos a las clases creadas clases PO (Page Object).

Una PO es una clase que representa una página web.

Creamos una clase PO para cada página html

ManagerPage

userName

newCustomer

EditCustomer

...

newCustomer()

editCustomer()

...

# CLASES LOGINPAGE Y MANAGERPAGE

P

OJO: esto es así en este ejemplo particular, ya que NO disponemos del código fuente de la aplicación.

- Estas clases estarán implementadas en src/main/java

```
public class LoginPage {  
    WebDriver driver;  
    WebElement userID;  
    WebElement password;  
    WebElement login;  
    WebElement pTitle;  
  
    public LoginPage(WebDriver driver){  
        this.driver = driver;  
        this.driver.get("http://demo.guru99.com/V4");  
        userID = driver.findElement(By.name("uid"));  
        password =  
            driver.findElement(By.name("password"));  
        login =  
            driver.findElement(By.name("btnLogin"));  
        pTitle =  
            driver.findElement(By.className("barone"));  
    }  
  
    public ManagerPage login(String user,  
                           String pass){  
        userID.sendKeys(user);  
        password.sendKeys(pass); servicio de login  
        login.click();  
        return new ManagerPage(driver);  
    }  
  
    public String getPageTitle(){  
        return pTitle.getText();  
    }  
}
```

```
public class ManagerPage {  
    WebDriver driver;  
    WebElement homePageUserName;  
    WebElement newCustomer;  
    ...  
    WebElement logOut;  
  
    public ManagerPage(WebDriver driver){  
        this.driver = driver;  
        homePageUserName =  
            driver.findElement(By.xpath("//table//  
                                tr[@class='heading3']"));  
        newCustomer =  
            driver.findElement(By.linkText("New Customer"));  
        logOut =  
            driver.findElement(By.linkText("Log out"));  
    }  
  
    public String getHomePageDashboardUserName(){  
        return homePageUserName.getText();  
    }  
    ...  
}
```



Las clases que representan cada una de las páginas HTML contienen código Webdriver y por lo tanto, dependen del código html de nuestra aplicación a probar.  
Los métodos de las PO implementan los **SERVICIOS** de la páginas HTML!!

# CLASE TEST LOGIN PAGE

- El test lo implementaremos en src/test/java

```
public class TestLoginPage {  
    WebDriver driver;  
    LoginPage poLogin;  
    ManagerPage poManagerPage;  
  
    @BeforeEach  
    public void setup(){  
        driver = new ChromeDriver();  
        poLogin = new LoginPage(driver);  
    }  
  
    @Test  
    public void test_Login_Correct(){  
  
        String loginPageTitle = poLogin.getLoginTitle();  
        Assertions.assertTrue(loginPageTitle.toLowerCase().contains("guru99 bank"));  
        poManagerPage = poLogin.login("mngr34733", "AbEvydU");  
  
        Assertions.assertTrue(poManagerPage.getHomePageDashboardUserName()  
            .toLowerCase().contains("manger id : mngr34733"));  
    }  
  
    @AfterEach  
    public void tearDown() {  
        driver.quit();  
    }  
}
```



El test **NO** contiene código webdriver, por lo tanto, es independiente del código html de nuestra aplicación a probar



Acuérdate de cerrar el navegador después de cada test

# P PAGEFACTORY (I) S

P

La clase PageFactory proporciona objetos de las clases PageObject. Para ello tendremos que:

- anotar los atributos de la clase PageObject con `@FindBy`,
- y utilizar el método estático `PageFactory.initElements()` en el test:  
`initElements(WebDriver driver, java.lang.Class PageObjectClass)`

El constructor de la clase PO requiere un parámetro de tipo Webdriver

```
public class LoginPage {  
    WebDriver driver;  
    @FindBy(name="uid") WebElement userID;  
    @FindBy(name="password") WebElement password;  
    @FindBy(name="btnLogin") WebElement login;  
    @FindBy(className="barone") WebElement loginTitle;  
  
    public LoginPage(WebDriver driver){  
        this.driver = driver;  
        this.driver.get("http://demo.guru99.com/V4");  
    }  
  
    public ManagerPage login(String user, String pass) {  
        userID.sendKeys(user);  
        password.sendKeys(pass);  
        login.click();  
  
        return PageFactory.initElements(driver,  
                                         ManagerPage.class);  
    }  
  
    public String getLoginTitle(){ ... }  
}
```



El método `initElements()` invoca al constructor de la clase.

La anotación `@FindBy` inyecta los valores de los atributos de forma "lazy", localizando cada elemento en el momento en el que vamos a usarlo.

```
public class ManagerPage {  
    WebDriver driver;  
    @FindBy(xpath="//table//tr[@class='heading3']")  
    WebElement userName;  
    @FindBy(linkText="New Customer")  
    WebElement newCustomer;  
    @FindBy(linkText="Log out")  
    WebElement logOut;  
  
    public ManagerPage(WebDriver driver){  
        this.driver = driver;  
    }  
    public String getUserName(){ ... }  
}
```

# PAGEFACTORY (II)

P

S

S

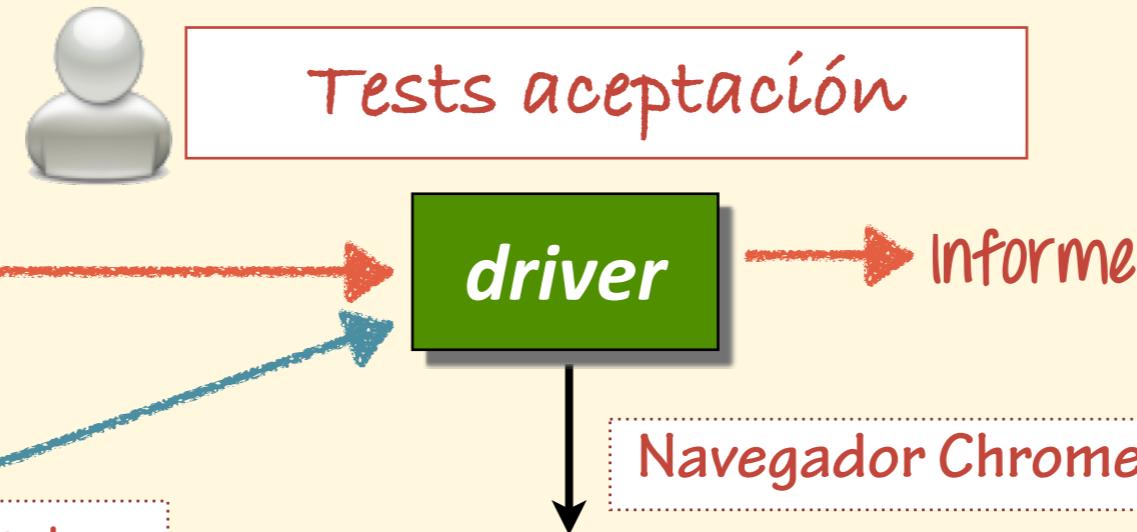
- Usaremos el método PageFactory.initElements() en nuestro test para crear una instancia de la clase que representa la página web de la aplicación a probar

```
public class TestLoginPage {  
    WebDriver driver;  
    LoginPage poLogin;  
    ManagerPage poManagerPage;  
  
    @BeforeEach  
    public void setup(){  
        driver = new ChromeDriver();  
        poLogin = PageFactory.initElements(driver, LoginPage.class);  
    }  
    @Test  
    public void test_Login_Correct(){  
        String loginpageTitle = poLogin.getLoginTitle();  
        Assertions.assertTrue(loginpageTitle.toLowerCase()  
                            .contains("guru99 bank"));  
        poManagerPage = poLogin.login("mngr34733", "AbEvydU");  
        Assertions.assertTrue(poManagerPage.getUserName()  
                            .toLowerCase()  
                            .contains("manger id : mngr34733"));  
    }  
    @AfterEach  
    public void tearDown() {  
        driver.quit();  
    }  
}
```

# Y AHORA VAMOS AL LABORATORIO...

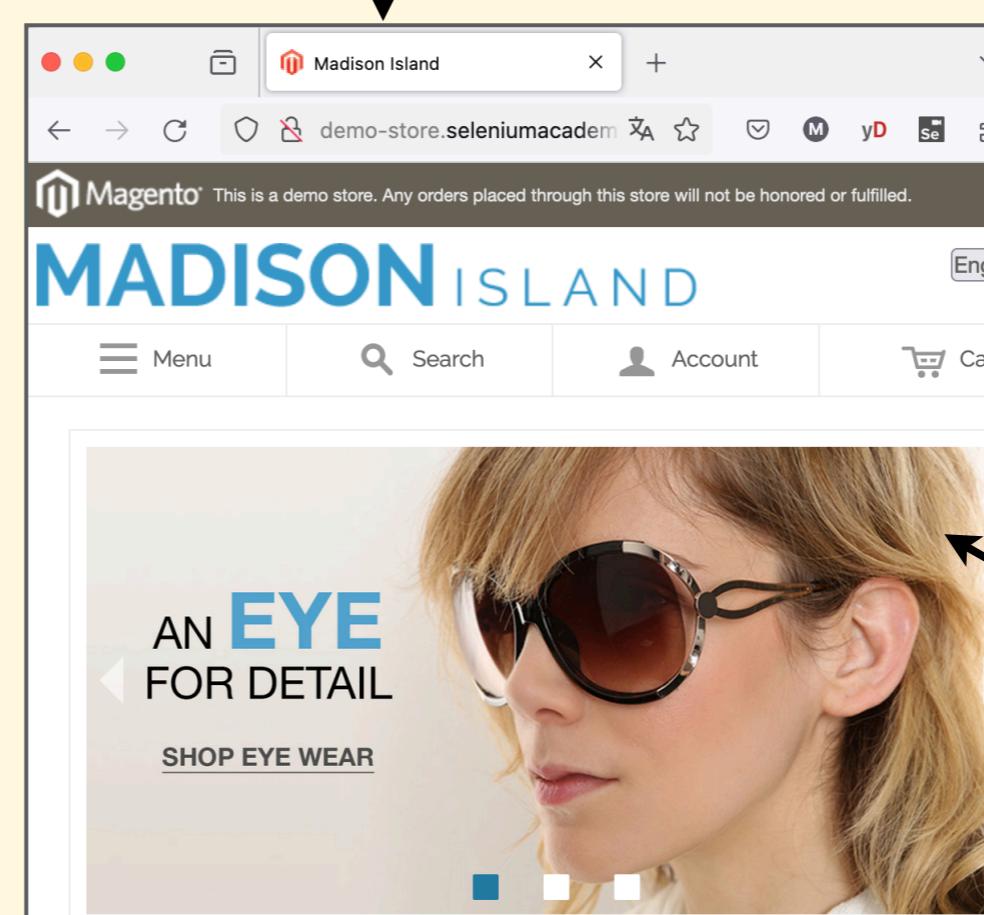
Vamos a implementar tests de aceptación (para validar propiedades emergentes funcionales) para una aplicación web con selenium WebDriver

Camino	Datos Entrada	Resultado Esperado
C1	d1=... d2=... ...	r1
..		
CM	d1=... d2=... ...	rM

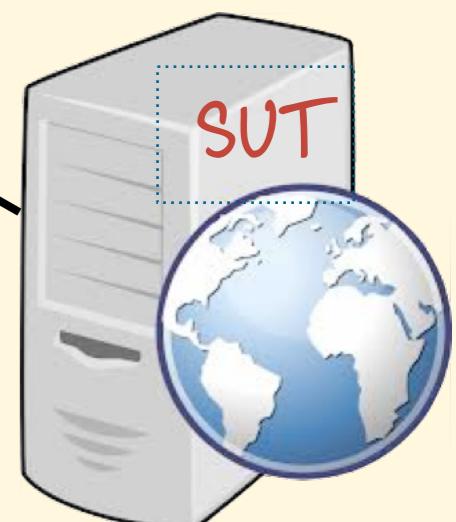


usaremos JUnit + el API de WebDriver para Chrome

Dado que no disponemos del código de la aplicación web a probar, usaremos un proyecto Maven que sólo va a contener el código de las pruebas de aceptación



Sevidor web



# REFERENCIAS BIBLIOGRÁFICAS

- Selenium WebDriver 3 Practical Guide. Unmesh Gundecha and Satya Avasarala. Packt Publishing. 2018
  - Capítulos 1,4 y 9
- Selenium design patterns and best practices : build a powerful, stable, and automated test suite using Selenium WebDriver. Dima Kovalenko, Jim Evans, Jeremy Segal. Packt Publishing, 2014
  - Capítulo 7: The Page Object Pattern
- Software testing and quality assurance. Kshirasagar Naik & Priyadarshi Tripathy. Wiley. 2008
  - Capítulo 14: Acceptance testing
- Software Engineering. 9th edition. Ian Sommerville. 2011
  - Capítulo 8.3: Release testing
- Tutorial Selenium (<http://www.guru99.com/selenium-tutorial.html>)