

PPSS PLANIFICACIÓN Y PRUEBAS DE SISTEMAS SOFTWARE

Curso 2021-22

Sesión S07: Pruebas del sistema y aceptación



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

Pruebas del sistema

- Objetivo: encontrar defectos en las funcionalidades (desde el punto de vista del desarrollador)
- Diseño:
 - Método de transición de estados
 - 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 requerimientos
 - Método basado en escenarios
- Automatización:
 - Selenium IDE (para aplicaciones con interfaz web)
 - Webdriver (para aplicaciones con interfaz web)
 - Se pueden usar también para pruebas del sistema

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

ACEPTACIÓN

SISTEMA

INTEGRACIÓN

UNIDADES

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!!!

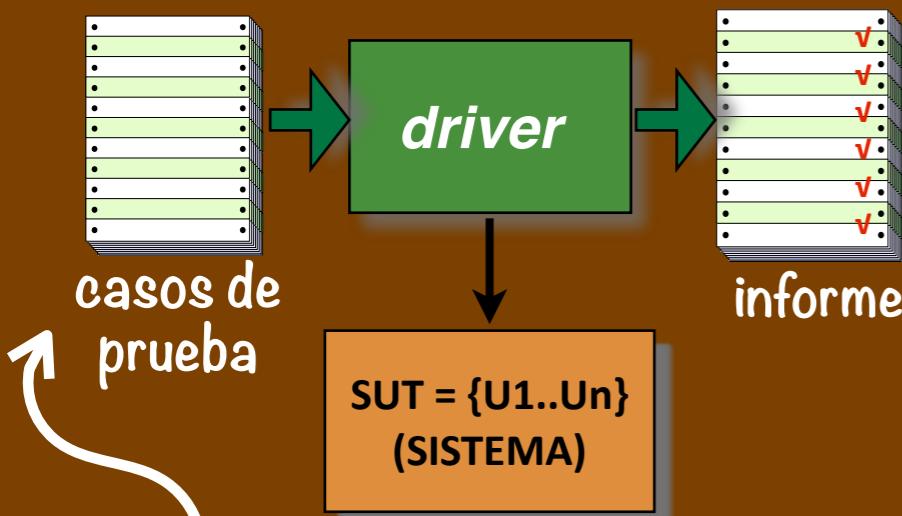


PS PR. SISTEMA VS PR. ACEPTACIÓN SP

VALIDACIÓN

VERIFICACIÓN

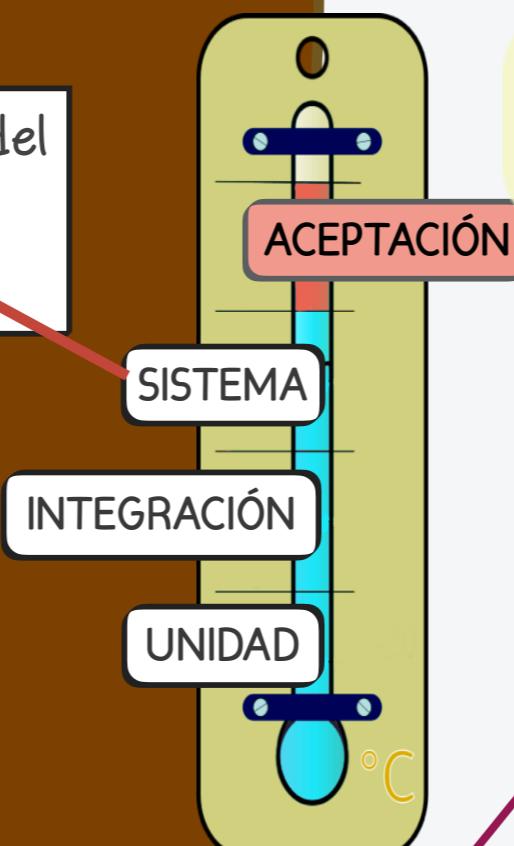
Objetivo: encontrar **DEFECTOS** derivados del **COMPORTAMIENTO** del sw como un todo
REQUIERE los **REQUISITOS** del sistema



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

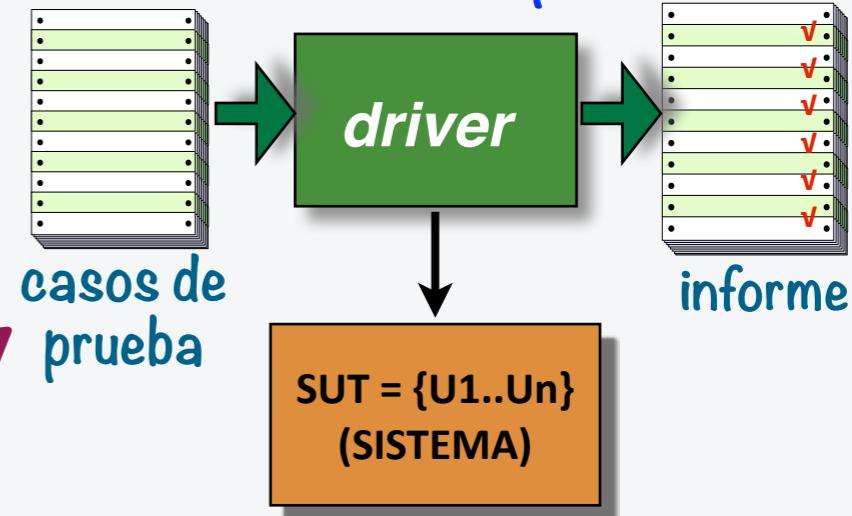


Los casos de prueba se diseñan desde el **punto de vista del USUARIO (CLIENTE)**

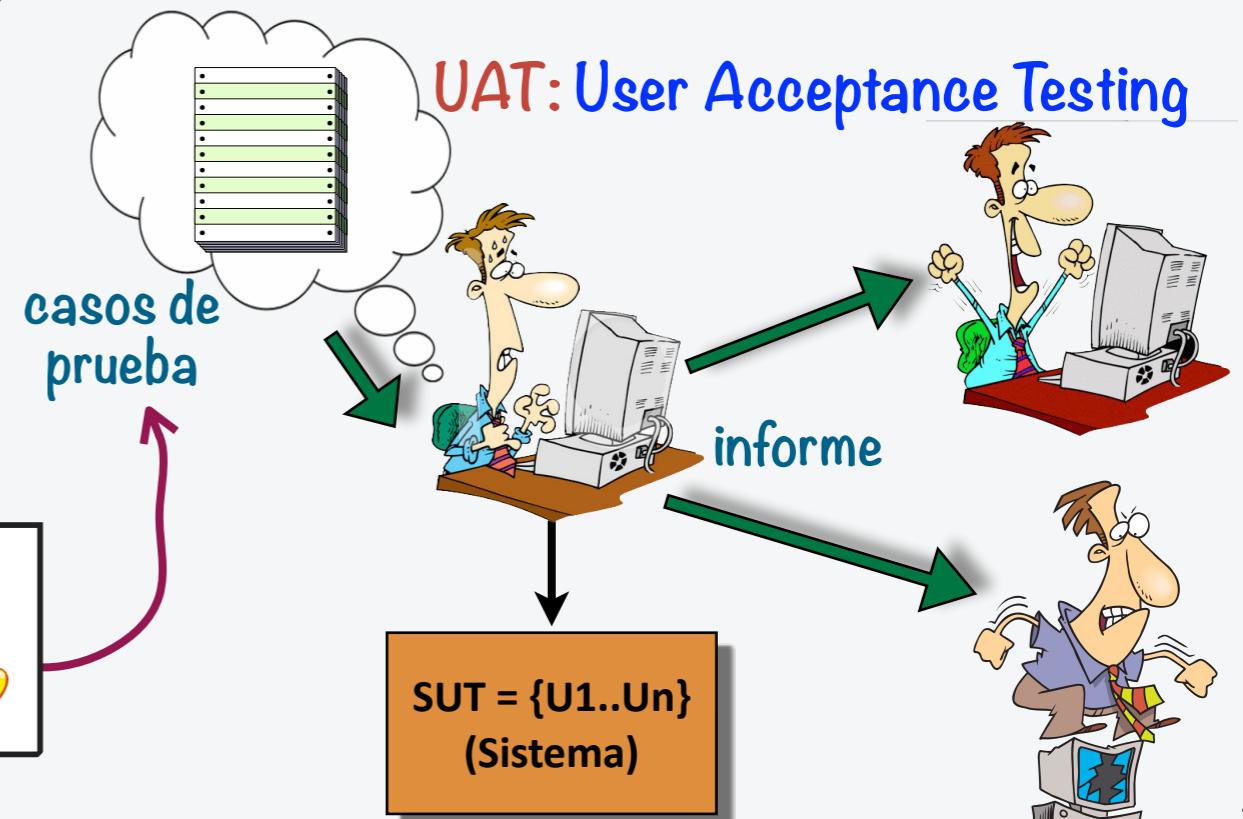


Objetivo: valorar en qué grado el software desarrollado satisface las expectativas del cliente
REQUIERE los **CRITERIOS DE ACEPTACIÓN**

BAT: Business Acceptance Testing



UAT: User Acceptance Testing



PRUEBAS DEL SISTEMA

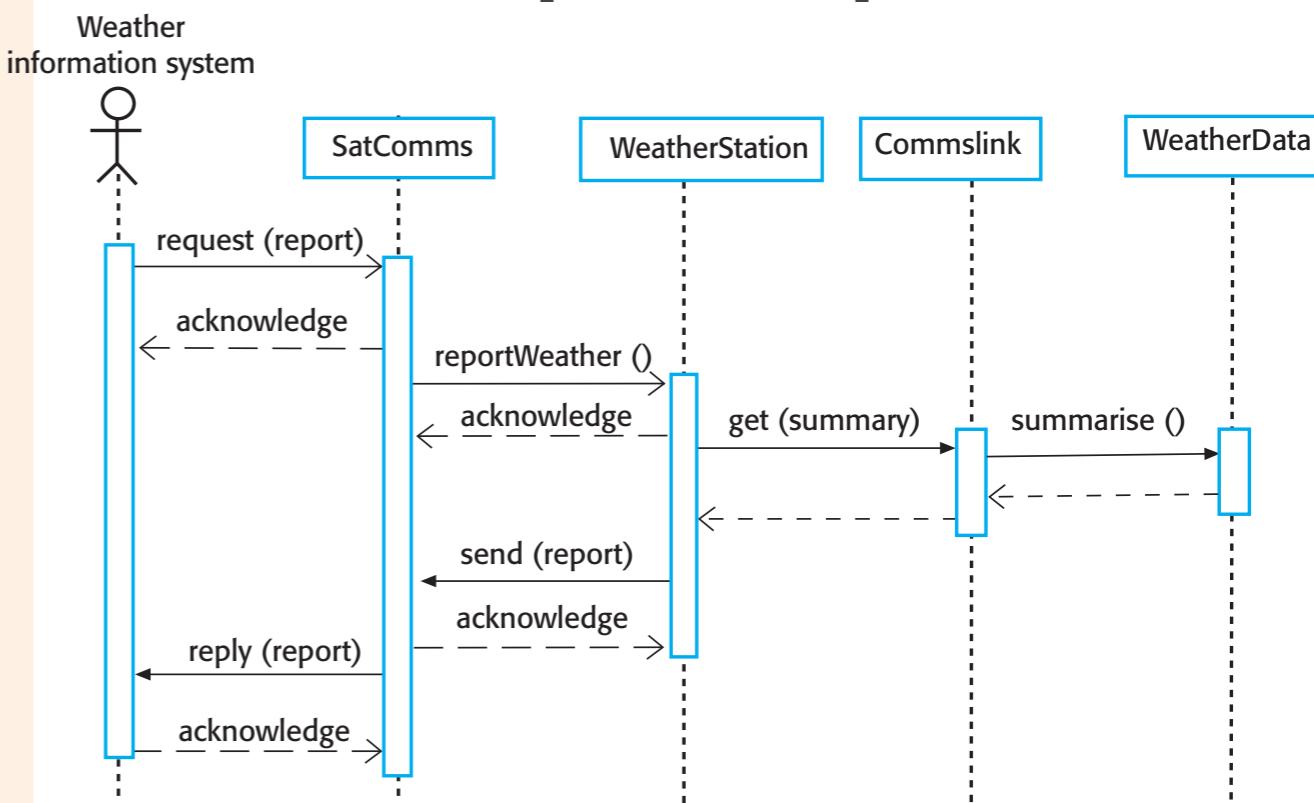


Recuerda que "vemos" el sistema desde el punto de vista del DESARROLLADOR!!

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



DISEÑO DE PRUEBAS DE SISTEMA

P

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

S

S

Método de diseño de **transición de estados**:

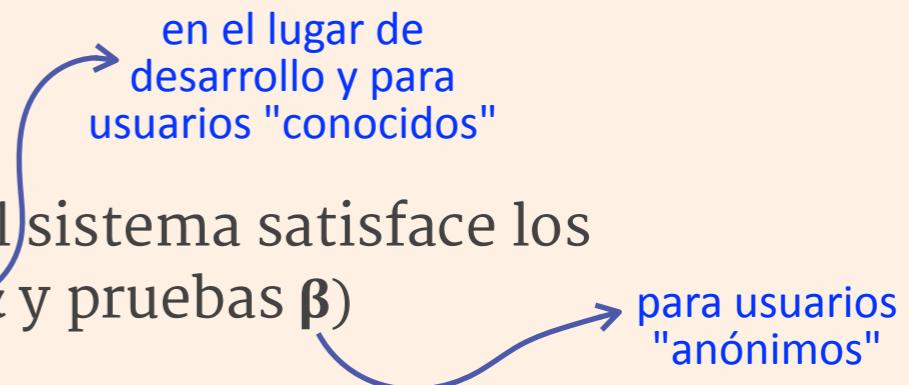
- Se usa en sistemas con ESTADO. Un estado viene dado por un conjunto de valores de variables del sistema.
 - Por ejemplo, un sistema de gestión de pedidos (en donde un pedido puede estar "iniciado", "cancelado", "procesado", "pendiente", ...)
- En este caso, a partir de la especificación se obtiene una representación en forma de grafo (denominado diagrama de transición de estados). Dicho grafo modela los estados de una entidad del sistema, representados en sus nodos. Las aristas representan las transiciones entre estados.
- A partir del grafo, se selecciona un conjunto de caminos mínimo para conseguir un objetivo concreto. Por ejemplo: recorrer todas las transiciones del grafo.
- Finalmente se determinan los datos de entrada y salida esperados para los comportamientos que ejercitan dicho conjunto de caminos

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 α y pruebas β)
 - **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
- Ejemplos de **métodos 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

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

DISEÑO DE PRUEBAS BASADO EN ESCENARIOS (I)

Capítulo 8.3.2 "Software Engineering" 9th. Ian Sommerville

- Un escenario es una descripción de un ejemplo de interacción del usuario/s con el sistema. Un escenario debería incluir:

- Una descripción de las asunciones iniciales, una descripción del flujo normal de eventos, y de situaciones excepcionales; y una descripción del estado final del sistema cuando el escenario termine

- Ejemplo de escenario:

Kate is a nurse who specializes in mental health care. One of her responsibilities is to visit patients at home to check that their treatment is effective and that they are not suffering from medication side effects.

On a day for home visits, Kate logs into the MHC-PMS and uses it to print her schedule of home visits for that day, along with summary information about the patients to be visited. She requests that the records for these patients be downloaded to her laptop. She is prompted for her key phrase to encrypt the records on the laptop.

One of the patients that she visits is Jim, who is being treated with medication for depression. Jim feels that the medication is helping him but believes that it has the side effect of keeping him awake at night. Kate looks up Jim's record and is prompted for her key phrase to decrypt the record. She checks the drug prescribed and queries its side effects. Sleeplessness is a known side effect so she notes the problem in Jim's record and suggests that he visits the clinic to have his medication changed. He agrees so Kate enters a prompt to call him when she gets back to the clinic to make an appointment with a physician. She ends the consultation and the system re-encrypts Jim's record.

After, finishing her consultations, Kate returns to the clinic and uploads the records of patients visited to the database. The system generates a call list for Kate of those patients who she has to contact for follow-up information and make clinic appointments.

DISEÑO DE PRUEBAS BASADO EN ESCENARIOS (II)



- El escenario anterior describe las siguientes características (requisitos o requerimientos) de nuestro sistema:
 - Authentication by logging on to the system.
 - Downloading and uploading of specified patient records to a laptop
 - Home visit scheduling
 - Encryption and decryption of patient records on a mobile device
 - Record retrieval and modification
 - Links with the drugs database that maintains side-effect information
 - The system for call prompting
- Las pruebas basadas en escenarios normalmente prueban varios requerimientos en un mismo escenario. Por ello, además de probar los requerimientos individuales, también estamos probando la combinación de varios de ellos
 - El diseño de pruebas resultante se obtiene agregando los casos de prueba que tengan en cuenta los requerimientos anteriores: el tester, cuando ejecuta este escenario, adopta el rol de Kate, y debe contemplar situaciones como introducir credenciales erróneas, o información incorrecta sobre el paciente

PRUEBAS DE PROP. EMERGENTES FUNCIONALES: SELENIUM

<https://www.selenium.dev>



■ Las pruebas de propiedades emergentes funcionales tienen como objetivo el comprobar que el sistema ofrece la **FUNCIONALIDAD** esperada por el cliente

- ◆ Se diseñan con técnicas de caja negra, y prueban la funcionalidad del sistema a través de la interfaz de usuario
 - ▶ Diseño de pruebas basado en requerimientos
 - ▶ Diseño de pruebas basado en escenarios

■ **Selenium** es un conjunto de herramientas de pruebas open-source para automatizar pruebas de propiedades emergentes funcionales sobre **aplicaciones Web** (o cualquier aplicación cuyo cliente sea el navegador)

- ◆ **Selenium WebDriver** (API): Permiten crear tests robustos, que pueden escalarse y distribuirse en diferentes entornos 
- ◆ **Selenium IDE** (extensión de un navegador: Firefox, Chrome). Permite crear scripts de pruebas utilizando la aplicación web tal y como un usuario haría normalmente: a través del navegador. 
- ◆ **Selenium Grid**: es un servidor proxy que permite ejecutar tests en paralelo usando múltiples máquinas y diferentes navegadores. 

P SELENIUM IDE

P

Nombre del proyecto

Un proyecto Selenium está formado por un conjunto de Tests Cases.

Podemos agrupar los Tests Cases (en una Test Suite). Nuestro proyecto puede estar formado por Tests Cases y/o Tests Suites

Podemos añadir/borrar/modificar los Test Cases y/o Tests Suites

Ventana de ayuda

Podemos mostrar:

- la "traza" de ejecución de los tests (Log),
- la documentación de referencia de los comandos Selenese (Reference)

Ejecución de los tests

Command	Target	Value
1 open	/	
2 set window size	1012x693	
3 click	linkText=ESTUDIOS	
4 click	css=#wrappe rContenidoM	

Project: Prueba*

Tests + > https://www.ua.es

TestCase1

TestCase2

Aquí se muestran todos los drivers (Test case) y/o Tests Suites

Command

Target

Value

Description

Log Reference

Running 'Prueba'

1. open on / OK
2. setWindowSize on 1012x693 OK
3. Trying to find linkText=ESTUDIOS... OK

Warning Element found with secondary locator css=#enlaceMenuHeader3 > a. To use it by default, update the test step to use it as the primary locator.

Gestor de proyectos Selenium
Para crear/abrir/guardar proyectos (*.side)

Se v 3.17.0

Botón de grabación

Genera comandos Selenese mientras se interacciona con el navegador ("graba" las acciones del usuario).

No todos los comandos pueden generarse de forma automática

Editor de scripts

Cada driver se "programa" como un script de comandos SELENESE. Podemos editar cualquier comando

Detalle de comandos

Se muestra el detalle de los valores para cada uno de los elementos (partes) de cada comando

P EJEMPLO S

Abrimos SeleniumIDE, fijamos <https://web.ua.es/dccia/>, como url base, y pulsamos el botón grabar

1+2. Accederemos directamente a <https://web.ua.es/dccia/>

3+4. Seleccionamos el texto "Horario" en la página, y con botón derecho seleccionamos el comando "verifyText ... "

5+6. Desplegamos el elemento "COMPONENTES" y Pinchamos sobre "PROFESORADO"

7. Pinchamos sobre "Alfonso Galipienso, María Isabel"

8+9. Seleccionamos el texto "2516" y con botón derecho "verifyText ... "

Selenium IDE - dccia

Project: dccia

Tests +

Search tests...

https://web.ua.es

	Command	Target	Value
1	✓ open	/dccia/	
2	✓ set window size	1210x842	
3	✓ click	css=.col-12:nth-child(2).divider-text-left	
4	✓ verify text	css=.col-12:nth-child(2).divider-text-left	Horario de secretaría:
5	✓ click	id=navbarDropdown4_2	
6	✓ click	linkText=Profesorado	
7	✓ click	linkText=Alfonso Galipienso	
8	✓ click	css=.bloque_datos:nth-child(4) li:nth-child(2)	
9	✓ verify text	css=.bloque_datos:nth-child(4) li:nth-child(2)	teléfono +34 965903400 x 2516

Log Reference

Running 'test1'

- 1. open on /dccia/ OK 16:12:01
- 2. setWindowSize on 1210x842 OK 16:12:01
- 3. ... 16:12:01
- 6. click on linkText=Profesorado OK 16:12:06
- 7. click on linkText=Alfonso Galipienso OK 16:12:07
- 8. click on css=.bloque_datos:nth-child(4) li:nth-child(2) OK 16:12:08
- 9. verifyText on css=.bloque_datos:nth-child(4) li:nth-child(2) with value teléfono +34 965903400 x 2516 OK 16:12:09

'test1' completed successfully 16:12:10

Traza de ejecución y resultado del test

CÓDIGO DE LOS DRIVERS (TESTS CASES)

Se guardan en formato JSON

	Command	Target	Value
1	✓ open	/dccia/	
2	✓ set window size	1210x842	
3	✓ click	css=.col-12:nth-child(2).divider-text-left	
4	✓ verify text	css=.col-12:nth-child(2).divider-text-left	Horario de secretaría:
5	✓ click	id=navbarDropdown4_2	

El driver se implementa como un script de comandos selenese.

Los comandos se generan automáticamente durante la grabación, pero también podemos generarlos de forma manual

```
{
  "id": "07c5ab58-d1f7-4097-acec-be0f6e977d77",
  "version": "2.0",
  "name": "dccia",
  "url": "https://web.ua.es",
  "tests": [
    {
      "id": "1a2827ff-97a9-4841-adfb-474758de52d1",
      "name": "test1",
      "commands": [
        {
          "id": "9a1ab826-a31a-42ce-856a-aa4e37e93a20",
          "comment": "",
          "command": "open",
          "target": "/dccia/",
          "targets": [],
          "value": ""
        },
        {
          "id": "029419c5-044a-4eb6-8b13-89fe0fd01e57",
          "comment": "",
          "command": "setWindowSize",
          "target": "1210x842",
          "targets": [],
          "value": ""
        },
        {
          "id": "e60fd4af-d659-45ab-9b56-e59aa1014fd7",
          "comment": "",
          "command": "click", ...
        }
      ]
    }
  ]
}
```

El proyecto se puede guardar en un fichero con extensión .side en formato Json.

La herramienta permite importar proyectos para su edición/modificación/ejecución

Para cada comando se almacena:

- su identificador (id)
- comentario (comment)
- target
- targets opcionales
- value

COMANDOS SELENESE

S COMMAND + [TARGET] + [VALUE]

- Un comando Selenese puede tener uno o dos parámetros:
 - **target**: hace referencia a un elemento HTML de la página a la que se accede. (p.ej "link" indica un hipernlace)
 - **value**: contiene un texto, patrón, o variable a introducir en un campo de texto o para seleccionar una opción de una lista de opciones
- Una secuencia de comandos Selenese forman un "test script" (caso de prueba). Una secuencia de tests scripts forman una test suite

actions

interactúan directamente con los elementos de la página.

P.ej. "click", "type"

Muchas actions pueden llevar el sufijo AndWait

control flow

alteran el flujo secuencial de ejecución de los comandos.

Pueden crear ramas **condicionales**.

P.ej: "if", "else"

O pueden crear **bucles**. P.ej: "times", "while"

HAY 4 TIPOS DE COMANDOS:

accessors

permiten almacenar valores en variables
P.ej. "storeTitle"

assertions

verifican que se cumple una determinada condición

Hay 3 tipos:

- **Assert**: detienen la ejecución si no se cumple
- **Verify**: se registra el fallo y continúa la ejecución
- **WaitFor**: espera a que se cumpla una condición antes de fallar (el tiempo de espera se indica en milisegundos)

EJEMPLOS DE COMANDOS SELENESE

open: abre una página usando una URL y espera a que se cargue

click: pulsa con el botón izquierdo del ratón sobre un elemento de la página

verify title/assert title: verifica el valor del título de una página

verify text: verifica que un texto esté presente en algún elemento de la página

verify element present: verifica que un elemento, definido por su etiqueta html esté presente en algún sitio de la página

store: almacena un string en una variable

store text: obtiene el texto de un elemento de la página y lo almacena en una variable

type: escribe un texto en un campo html

SINTAXIS:

open url

click locator

verify title text, assert title text

verify text locator text

verify element present locator

store text variableName

store text locator variableName

type locator value

Ver <https://www.seleniumhq.org/selenium-ide/docs/en/api/commands/>

Ver <https://www.seleniumhq.org/selenium-ide/docs/en/introduction/control-flow/>

Ver <https://www.guru99.com/first-selenium-test-script.html>

LOCATORS

Ver <https://www.guru99.com/locators-in-selenium-ide.html>

- Se utilizan en el campo **target** e identifican un elemento en el código de una página web (un botón, un cuadro de texto...). La sintaxis es **locatorType=location**

- Podemos utilizar los siguientes tipos de "locators":
 - id**: hace referencia al atributo **id** del elemento html
 - name**: atributo **name** del elemento html. Adicionalmente podemos añadir un "filtro" consistente en añadir otro atributo adicional junto con su valor
 - xpath**: es el lenguaje utilizado para localizar nodos en un documento HTML
 - * Podemos utilizar rutas absolutas: `xpath=/html/body/form[1]`
 - * o rutas relativas: `xpath=/form[1]`

Command	Target	Value
open	http://demo.guru99.com/test/newtours/	
verifyText	//tr[4]/td/table/tbody/tr[2]/td/font	User*
verifyElementPresent	name=password	
open	http://demo.guru99.com/test/facebook.html	
type	id=email	hola

EJEMPLOS DE LOCATORS (I)

- Suponemos que el código HTML de nuestra página web es:

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

Usamos un filtro para
refinar la búsqueda

- id=loginForm (línea 3)
- name=username (4)
- name=continue type=button (7)
- xpath=/html/body/form[1] (3) es una ruta absoluta
- //form[1] (3) es una ruta relativa (encuentra el primer "form" en el HTML)

- linkText=Continue (4)
- linkText=Cancel (5)

```
1 <html>
2   <body>
3     <p>Are you sure you want to do this?</p>
4     <a href="continue.html">Continue</a>
5     <a href="cancel.html">Cancel</a>
6   </body>
7 <html>
```

LOCALIZACIÓN DE ELEMENTOS DESDE CHROME

- Podemos obtener información de los elementos de una página web mediante la utilidad "inspeccionar" desde dicha página web con el menú contextual (desde dicho elemento)

1. Situamos el ratón dentro del cuadro de texto y seleccionamos "Inspecionar" desde el menú contextual (botón derecho)

2. En la ventana del inspector se resalta el código del elemento

3. Si nos posicionamos sobre el resultado en azul de la ventana de inspección (2), también se resaltará en elemento en la página

4. En cualquier momento podemos "buscar" cualquier otro elemento activando el botón de inspección

The screenshot shows a travel website for Aruba. A red box highlights the 'User Name:' input field. A red arrow points from this field to a red box containing step 3. Another red arrow points from the same input field to a red box containing step 4. A third red arrow points from the input field to the 'Elements' tab of the developer tools sidebar, which contains the HTML code for the input field. A fourth red arrow points from the 'Elements' tab to a red box containing step 2. The developer tools sidebar also shows other parts of the page's HTML structure.

LOCALIZACIÓN DE ELEMENTOS DESDE SELENIUM IDE

P

Project: Prueba*

Tests + DΞ ▷ ⏺ ⏴

P Search tests...

DemoTours*

Test 1

Test 2

1. elegimos el comando

Command

Target

Value

Description

2. Activamos la selección del target en la página



4. Se "rellena" al seleccionar el elemento en la página

Log • Reference

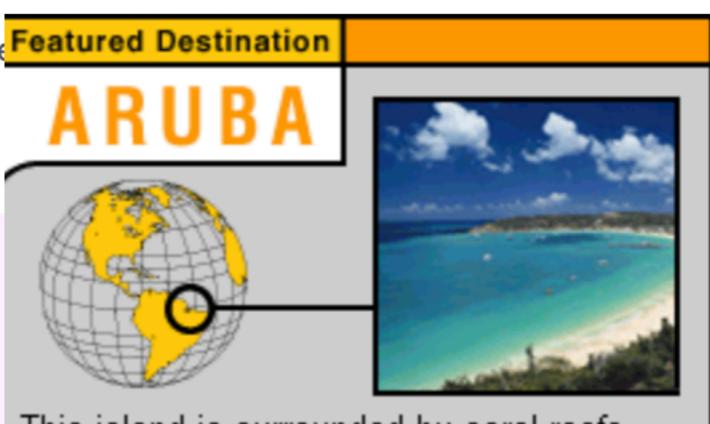
verify element present locator

Soft assert that the specified element is somewhere on the page. The test will continue.

arguments:

locator - An element locator.

Aquí podemos consultar la información sobre el comando



5. También podemos buscar en la página el valor de target. Se resaltará momentáneamente el elemento en la página

3. Al pasar el ratón por la página se resaltan los elementos. Seleccionamos el elemento que nos interese

Jul 6, 2017

Find A Flight

Registered users can sign-in here to find the lowest fare on participating airlines.

User Name:

Password:

CONTROL FLOW

P

Command	Target	Value
execute script	return "a"	myVar
if	<code> \${myVar} === "a"</code>	
execute script	return "a"	output
else if	<code> \${myVar} === "b"</code>	
execute script	return "b"	output
else		
execute script	return "c"	output
end		
assert	output	a

COMANDOS que controlan el flujo de ejecución:

- if, else if, else, end
- times, end
- do, repeat if
- while, end



<https://www.seleniumhq.org/selenium-ide/docs/en/introduction/control-flow/>

<http://www.cheat-sheets.org/saved-copy/jsquick.pdf>

COMANDO execute script

Parámetros: (sentencia javascript, variable)
Almacena en una variable el resultado de la sentencia o expresión javascript

EJEMPLOS de sentencias Javascript

`return "a"` → devuelve el string "a"

`${myVar} === "a"` → compara el valor de la variable myVar con el string "a"

COMANDO assert

Parámetros: (actual value, expected value)

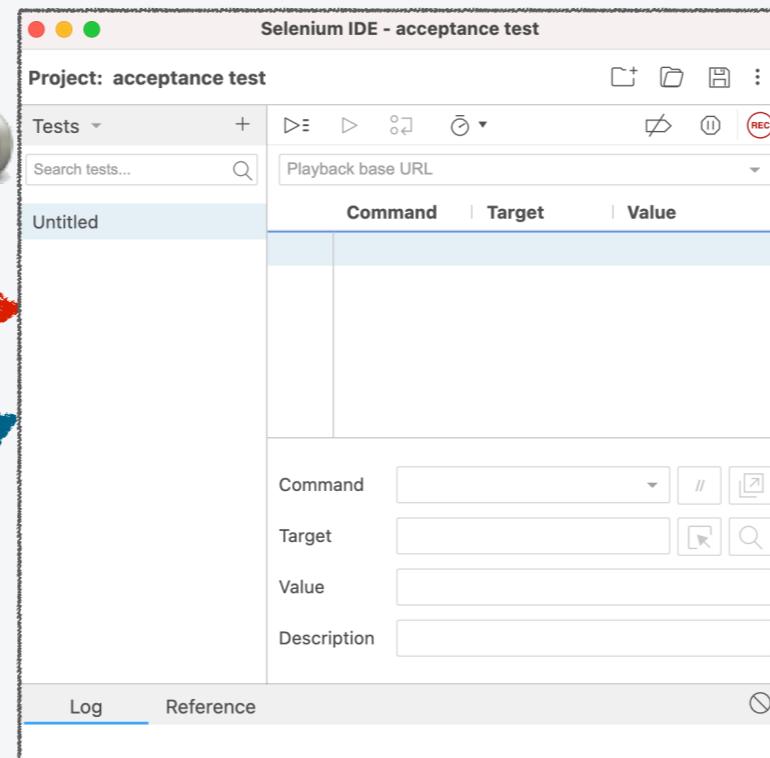
Convierte el valor de las variables en un string y compara sus valores. Si no coinciden el tests falla

Command	Target	Value
execute script	return 1	check
do		
execute script	return \${check} +1	check
repeat if	<code> \${check} < 3</code>	
assert	check	3

Y AHORA VAMOS AL LABORATORIO...

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

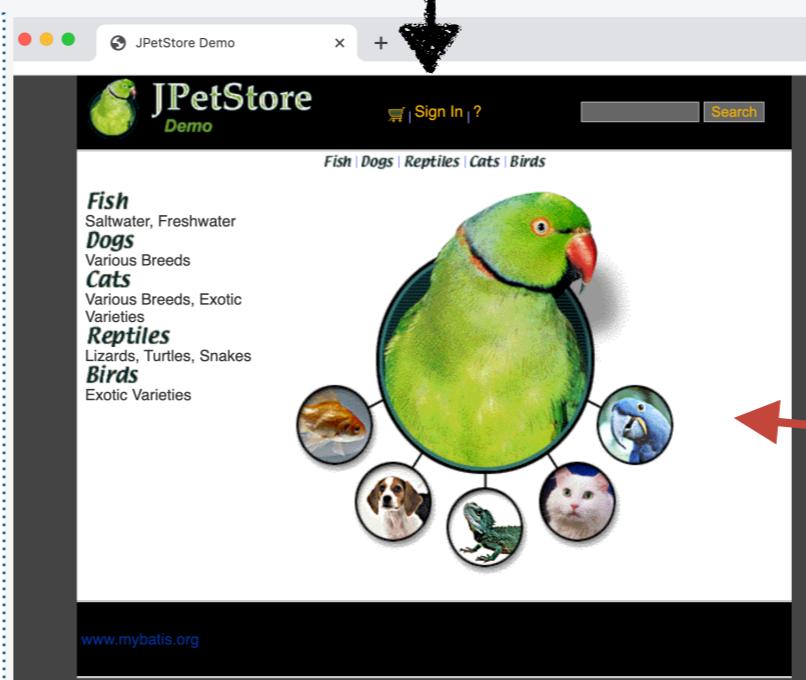
Camino	Datos Entrada	Resultado Esperado
C1	d1=... d2=... ...	r1
..		
CM	d1=... d2=... ...	rM



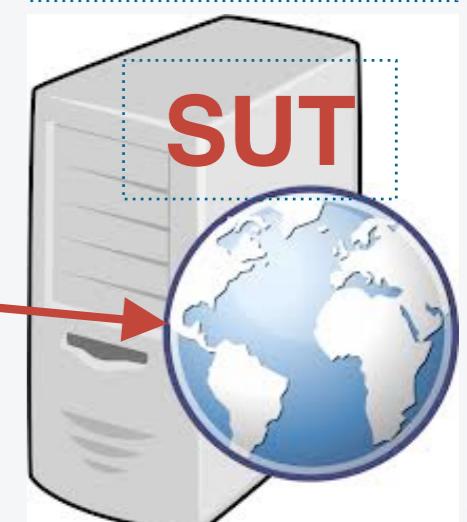
Tests aceptación

usaremos scripts Selenese
utilizando Selenium IDE

Navegador Chrome



Servidor web



P REFERENCIAS BIBLIOGRÁFICAS

- P
 - 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>)
 - Apartados 1..6