2.3 – Pruebas de Servicios de Internet

Tema 1 – Introducción a Pruebas de Software

Universidad Rey Juan Carlos

Micael Gallego
micael.gallego@urjc.es
@micael_gallego

Francisco Gortázar
francisco.gortazar@urjc.es
@fgortazar

code URJC

Introducción

- **Tipos de pruebas**
- Metodologías
- JUnit 5
- Conclusiones

Tipos de pruebas

- Existen **muchos tipos de pruebas**
  - Las pruebas se pueden clasificar atendiendo a **diferentes criterios** (tamaño, quién las crea, qué prueban…)
  - No hay una **taxonomía de pruebas globalmente aceptada.**
  - El **mismo nombre** se puede usar por **colectivos diferentes** para nombrar **diferentes tipos de pruebas**
  - Se presentarán los **tipos de pruebas más aceptados** indicando las ambigüedades cuando existan

Tipos de pruebas

- Existen muchas formas de clasificar las pruebas:
  - **Qué características prueban:** Funcionales o no funcionales
  - **Qué es el SUT:** Unidad, componente, integración, sistema
  - **Cómo se ejecutan:** Manuales o automáticas
  - **Con qué objetivo se hace la prueba:** Aceptación, smoke (humo), sanity (sanidad)
  - **Con qué conocimientos se diseñan:** Caja blanca frente a Caja negra

Tipos de pruebas

- Ya habeis visto pruebas unitarias
- ¿Qué vamos a ver aquí?
  - Pruebas Funcionales y no funcionales
  - Pruebas de integración y de sistema
  - Pruebas de aceptación
  - Todas automáticas

Tipos de pruebas

- Existen muchas formas de clasificar las pruebas:
  - **Qué características prueban:** Funcionales o no funcionales
  - **Qué es el SUT:** Unidad, componente, integración, sistema
  - **Cómo se ejecutan:** Manuales o automáticas
  - **Con qué objetivo se hace la prueba:** Aceptación, smoke (humo), sanity (sanidad)
  - **Con qué conocimientos se diseñan:** Caja blanca frente a Caja negra

Qué características prueban

- **Pruebas Funcionales:**
  - Pruebas que verifican la funcionalidad ofrecida por el SUT.
  - Comprueban que el SUT, partiendo de una situación determinada, cuando se interactúa con él ofrece los resultados esperados
  - **Ejemplo de prueba funcional:**
    - **Given:** En una tienda de comercio electrónico, si el usuario está en la página de un producto y tiene la cesta vacía
    - **When:** El usuario pulsa el botón de añadir producto a la cesta
    - **Then:** El icono de la cesta aparece un 1 producto

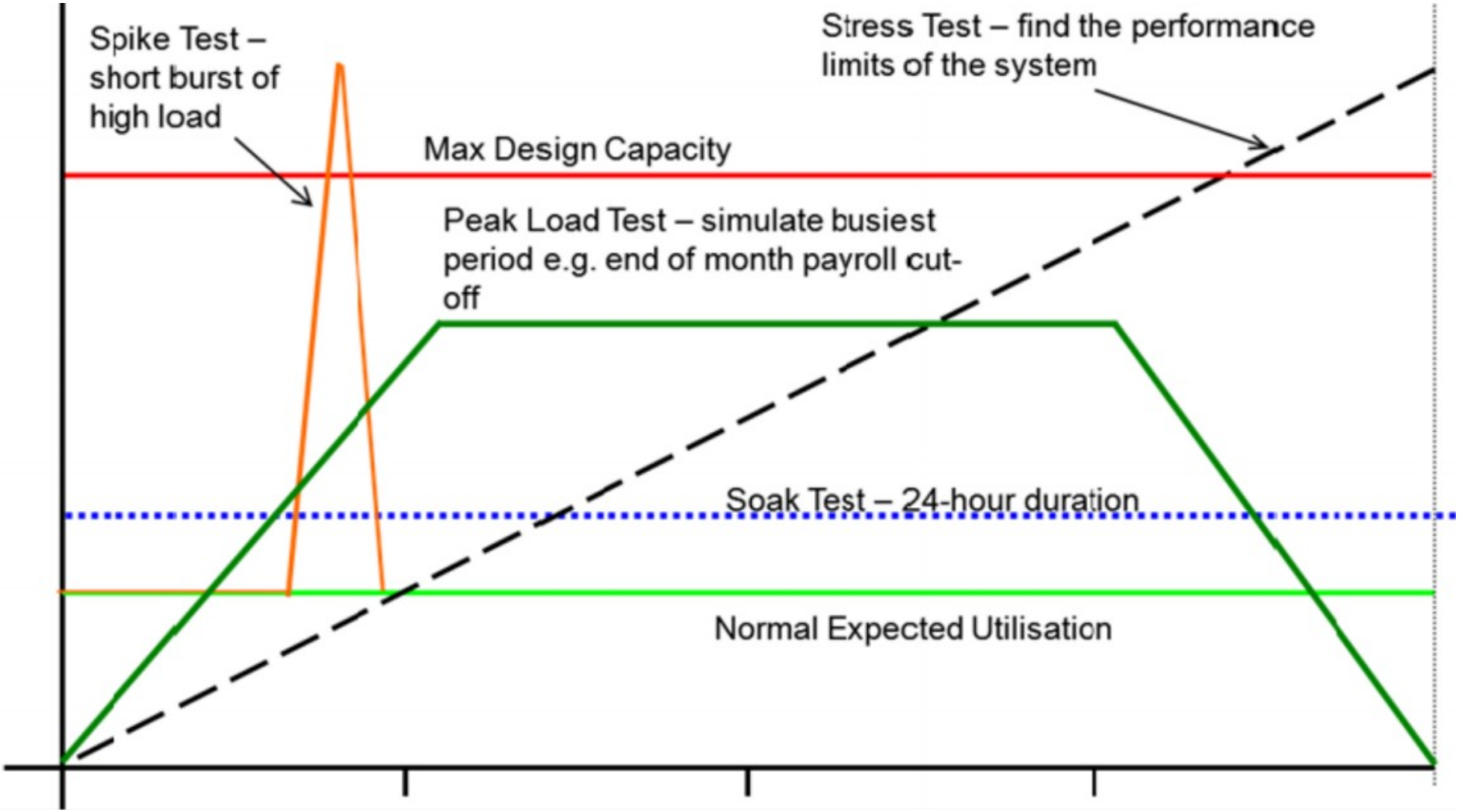
Pruebas No Funcionales

- **Pruebas de Rendimiento:** verifican que el SUT cumple con el rendimiento esperado que se puede medir en función del número de transacciones por segundo, tiempo máximo de generación de respuesta, etc.
- **Prueba de carga o de esfuerzo o escalabilidad:** verifican que el SUT cumple con el comportamiento y rendimiento esperado bajo una pesada carga de los datos, la repetición de ciertas acciones de los datos de entrada, los grandes valores numéricos, consultas grandes a una base de datos o para comprobar el nivel de los usuarios concurrentes

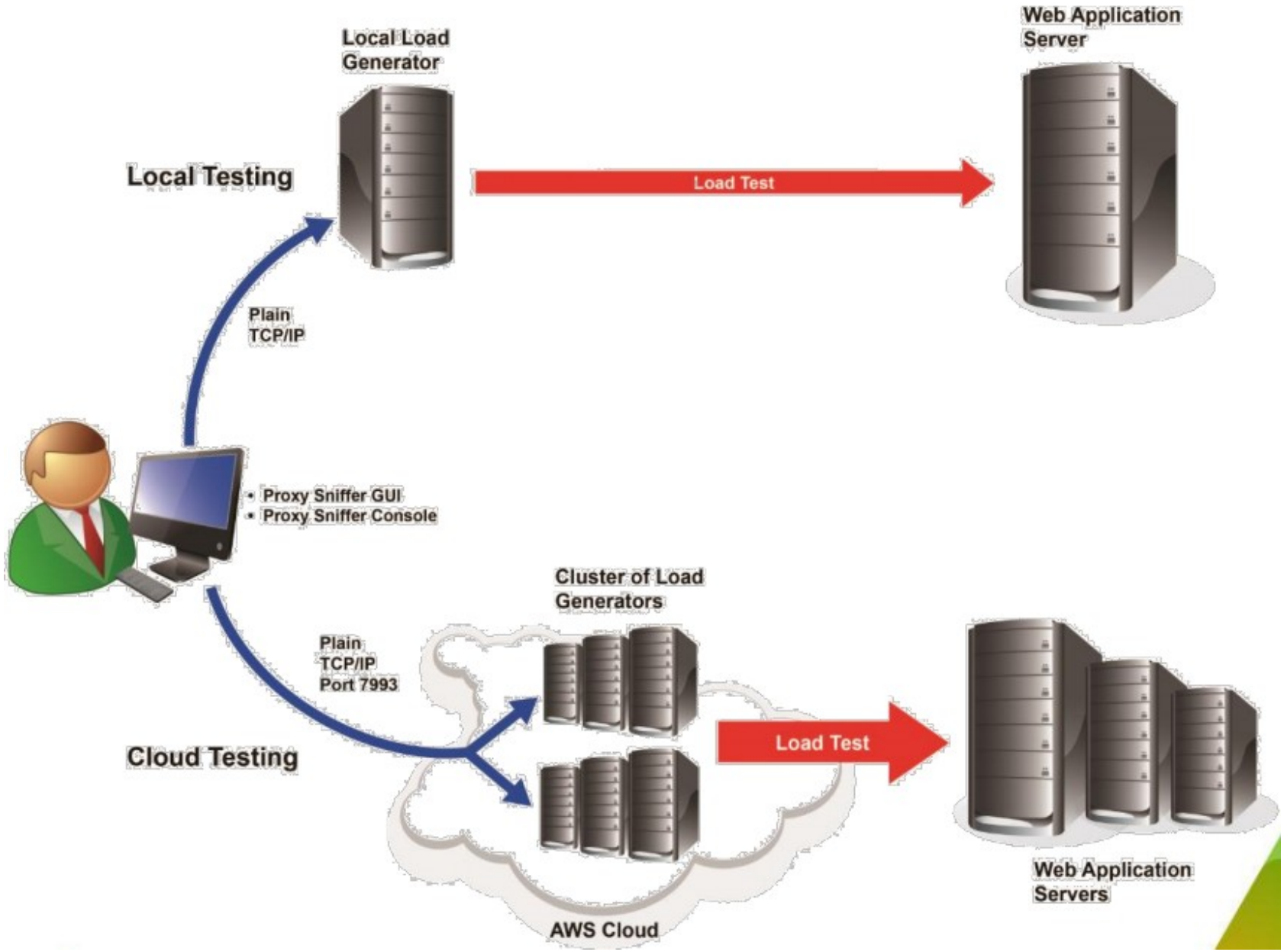
Pruebas No Funcionales

- **Prueba de estabilidad:** es una prueba que se ejecuta durante un tiempo largo y que busca anomalías como pérdidas de memoria u otras degradaciones por la continuidad de la ejecución
- **Pruebas de estrés o volumen:** donde se escala la cantidad de carga con el tiempo hasta que se encuentren los límites del sistema; con el objetivo de examinar cómo falla y vuelve a su funcionamiento normal.

Pruebas No Funcionales



Pruebas No Funcionales



Pruebas No Funcionales

code URJC

- **Pruebas de seguridad**
  - SAST
    - Detectan malas prácticas que pueden llevar a problemas de seguridad en el software
    - Detección de dependencias con vulnerabilidades
  - IAST
    - Analizan el tráfico de la aplicación en busca de posibles problemas (CORS, autenticación en claro...)
  - DAST o Pentesting
    - Realizan ataques intentanto ganar acceso al sistema o provocar denegación de servicio
    - Normalmente realizan un análisis IAST previo
  - En producción
    - Filtrar peticiones sospechosas, elevar alarmas y auditarlas
    - WAF: firewall externo a la aplicación
    - RAST: librería que se embebe en la aplicación

Pruebas No Funcionales

- **Otros tipos:**
  - Pruebas de usabilidad
  - Pruebas de manejo y Recuperacion de Errores y desastres
  - Pruebas de instalación y desinstalación
  - Pruebas de configuración, compatibilidad o portabilidad
  - Pruebas de internacionalización y Localización
  - Pruebas de manejo de fecha y hora

Pruebas No Funcionales

- **Herramientas**
  - Específicas de cada tipo de test no funcional

Pruebas de rendimiento, carga estrés y volumen de sistema

JMH

Java Microbenchmarking Harness

Pruebas de rendimiento unitarios y de integración

OWASP ZAP

Pruebas de seguridad de aplicaciones web

Tipos de pruebas

- Existen muchas formas de clasificar las pruebas:
  - **Qué características prueban:** Funcionales o no funcionales
  - **Qué es el SUT:** Unidad, componente, integración, sistema
  - **Cómo se ejecutan:** Manuales o automáticas
  - **Con qué objetivo se hace la prueba:** Aceptación, smoke (humo), sanity (sanidad)
  - **Con qué conocimientos se diseñan:** Caja blanca frente a Caja negra

Qué es el SUT

- **Pruebas de sistema**
  - Las pruebas permiten verificar si el **software se comporta como se espera**.
  - Implícitamente estamos asumiendo que se **prueba el sistema completo**, que es sobre el que se definen los requisitos.
  - En estas pruebas el SUT (Sujeto bajo prueba) es el **sistema completo**
  - A veces se las llama **pruebas extremo a extremo**, ***end to end*** o **e2e** (aunque existen ciertas diferencias)

Qué es el SUT


xUnit Test Patterns
Micael
Secure | https://www.amazon.es/xUnit-Test-Patterns-Refactoring-Signature/dp/0131495054/ref=sr_1_1?ie=UTF8&qid=1514398060&sr=8-1&k...
amazon.es prime
Libros en idiomas extranjeros
xunit testing software book
Compra hasta el 3 y recíbelo para Reyes con Envío 1 día GRATIS
Enviar a Micael Móstoles 28933
Todos los departamentos
Amazon.es de Micael
Ofertas
Cheques regalo
Vender
Ayuda
Hola Micael Cuenta y listas
Pedidos
Mi Prime
Cesta
Libros en idiomas extranjeros
Búsqueda avanzada
Todos los géneros
Preventa
Los más vendidos
Todos los Libros
Catalán
Gallego
Euskera
Inglés
xUnit Test Patterns: Refactoring Test Code (Addison-Wesle... y más de 950.000 libros están disponibles para Amazon Kindle . Más información
Volver a los resultados de la búsqueda de "xunit testing software book"
Echa un vistazo
xUnit Test Patterns: Refactoring Test Code (Addison Wesley Signature)
(Inglés) Tapa dura – jun 2007
de Gerard Meszaros (Autor)
31 opiniones de EE. UU.
Ver los 2 formatos y ediciones
Versión Kindle EUR 35,37
Tapa dura EUR 47,16 prime
Leer con nuestra App gratuita
3 Usado desde EUR 76,02
5 Nuevo desde EUR 47,16
Recíbelo antes de Reyes Magos.
¿Quieres recibirlo el viernes 29 dic.? Cómpralo antes de 22 hrs y 36 mins y elige Envío 1 día al completar tu pedido. Ver detalles
Automated testing is a cornerstone of agile development. An effective testing strategy will deliver new functionality more aggressively, accelerate user feedback, and improve quality. However, for many developers, creating effective automated tests is a unique and unfamiliar challenge.xUnit Test Patterns is the definitive guide to writing automated tests using xUnit, the most popular unit testing framework in use today. Agile coach and test automation expert Gerard Meszaros describes 68 proven patterns for
Leer más
Avisar de alguna información del producto errónea.
Ver las 3 imágenes
Compartir <Incrustar>
EUR 47,16 prime
Precio recomendado: EUR 55,05
Ahorras: EUR 7,89 (14%)
Precio final del producto
En stock.
Vendido y enviado por Amazon. Se puede envolver para regalo.
Cantidad: 1
Añadir a la cesta
Activar el pedido en 1-Clic
Enviar a Micael - Móstoles 28933
Añadir a la Lista de deseos
¿Tienes uno para vender?
Vender en Amazon
Comprados juntos habitualmente

Qué es el SUT

- **Pruebas de sistema: Limitaciones**
  - **Costosas de implementar:** probar un interfaz de usuario o una web simulando un usuario es complejo. Hay que tener en cuenta animaciones, selección de elementos en la interfaz, esperas, etc... La verificación de que el resultado es el esperado puede ser compleja (analizar un PDF...)
  - **Costosas de ejecutar:** Los sistemas son cada vez más complejos y necesitan más elementos: Base de datos, servidor web, sistemas externos, navegador web, etc... y ejecutar las pruebas en estos sistemas consume tiempo y recursos computacionales

Qué es el SUT

- **Pruebas de sistema: Limitaciones**
  - **Frágiles:** Un cambio en la interfaz de usuario implica el cambio de todos los tests involucrados.
  - **Poco flexibles:** Definir diferentes situaciones es complicado. Hay que generar conjuntos de datos con múltiples estados. Simular condiciones reales es complicado (por ejemplo: falta de conectividad con otros sistemas).

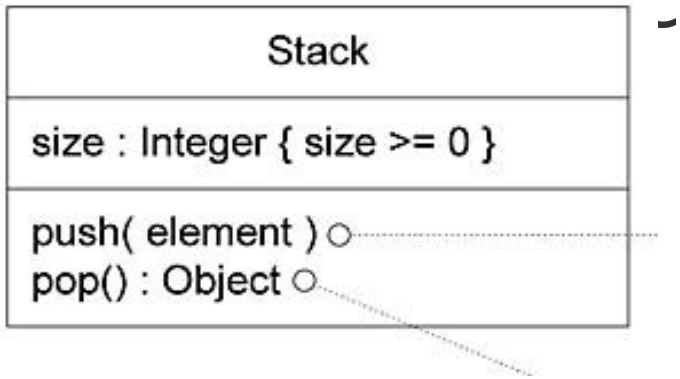
Para mitigar estos problemas, el software puede probarse

https://martinfowler.com/bliki/TestPyramid.html

por partes

Qué es el SUT

- **Pruebas unitarias**
  - **Históricamente** las pruebas unitarias son aquellas en las que el SUT es un **elementos básicos del software:** un método, una función, una clase, un módulo...



Qué es el SUT

- **Pruebas unitarias**
  - Nos focalizamos en probar una clase o método, consiguiendo **probar la mayoría de las situaciones posibles**
  - El objetivo es que **su ejecución sea rápida** para que se puedan ejecutar **frecuentemente**
  - Cuando se trabaja en una parte del código, las pruebas unitarias relacionadas con esa parte deberían poder **ejecutarse en menos de 10 segundos**

Qué es el SUT

- **Pruebas unitarias (Nueva definición)**
  - Algunos autores como **Rober C. Martin** y **Martin Fowler** consideran que los tests unitarios no son aquellos en los que el SUT es el elemento básico del software (clase, función…)
  - Amplían el concepto a tests creados por el **programador** para asegurarse de el **código hace lo que se espera** que haga, aunque **involucre varias clases colaborando entre sí**
  - Para ellos, los tests unitarios son aquellos usados por el programador para recibir **feedback rápido del código**
  - En general se asume que si el SUT no requiere llamadas a servicios externos a través de la red, el test es unitario

http://blog.cleancoder.com/uncle-bob/2017/05/05/TestDefinitions.html
https://martinfowler.com/bliki/UnitTest.html

Qué es el SUT

- **Pruebas unitarias**
  - Su objetivo es que **su ejecución sea rápida** para que se puedan ejecutar **frecuentemente**
  - Cuando los DOCs acceden a otros **sistemas vía red** o **acceden a disco**, se sustituyen por **dobles**
  - Hay que guardar el equilibrio entre:
    - Complejidad de implementación de tests
    - Velocidad de ejecución
    - Cubrir la mayor parte de las situaciones posibles durante su ejecución

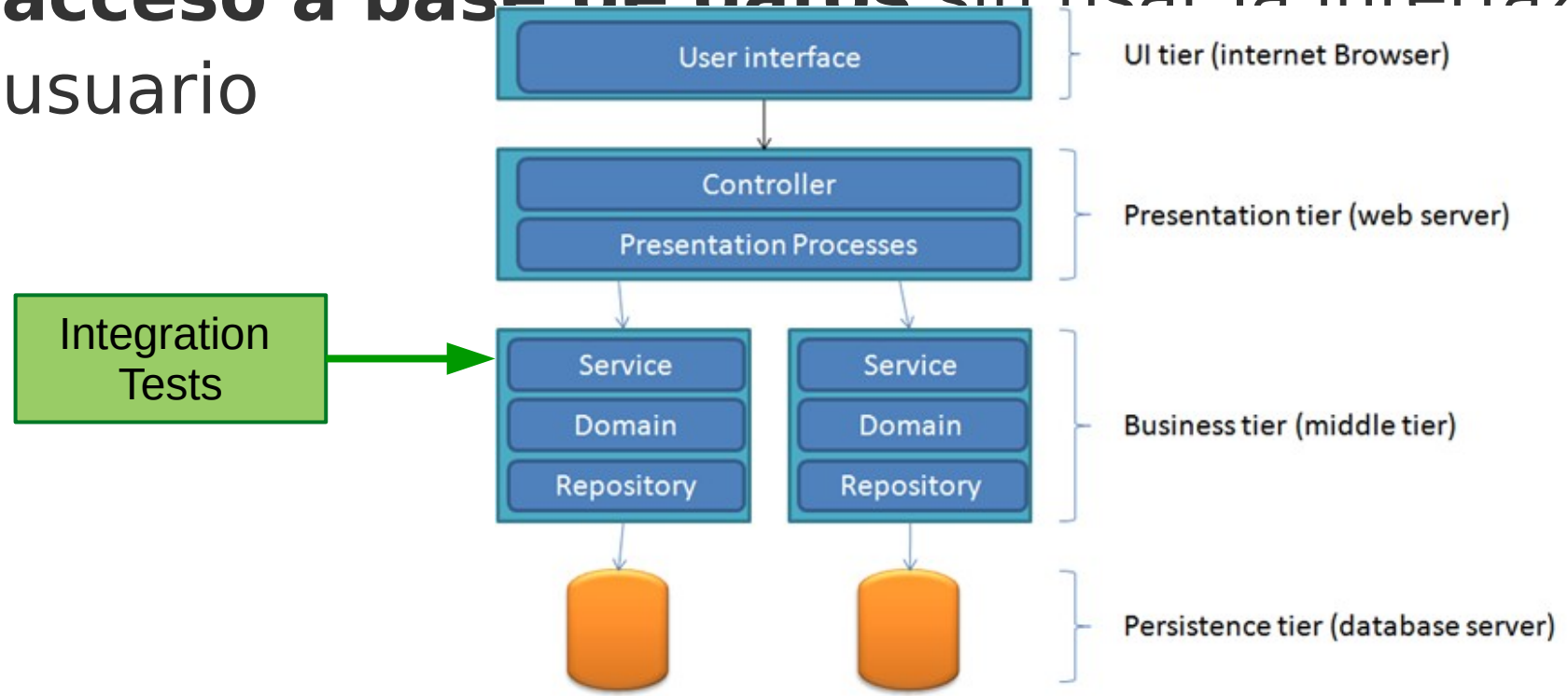
Qué es el SUT

code URJC

- **Pruebas de integración**
  - Históricamente las pruebas de integración eran aquellas en las que se veían **involucradas más de una clase**
  - Actualmente se llama pruebas de integración a aquellas en las que el SUT es un conjunto de **sub-sistemas de la aplicación interactuando entre sí**
  - El objetivo de estas pruebas consiste en **verificar que la comunicación entre los sub-sistemas** sea correcta
  - Habitualmente requiere la **comunicación** de diversos sistemas mediante **protocolos de red** o el uso de **librerías de terceros**

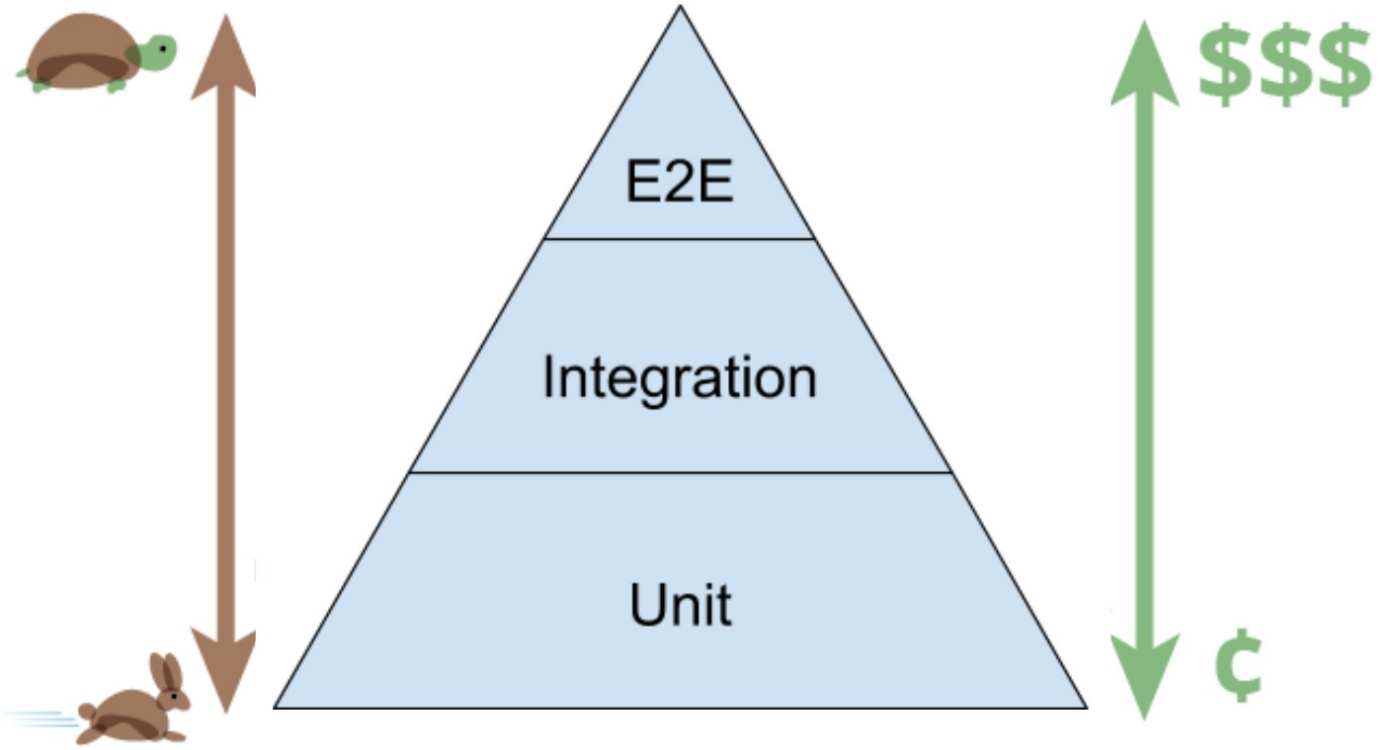
Qué es el SUT

- **Pruebas de integración**
  - Una prueba de integración puede ser probar la **lógica de negocio junto con la capa de acceso a base de datos** sin usar la interfaz de usuario



Qué es el SUT

- **Pirámide de tests**



- Google recomienda **70% Unitarios, 20% integración, 10% de sistema**

https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html

Tipos de pruebas

- Existen muchas formas de clasificar las pruebas:
  - **Qué características prueban:** Funcionales o no funcionales
  - **Qué es el SUT:** Unidad, componente, integración, sistema
  - **Cómo se ejecutan:** Manuales o automáticas
  - **Con qué objetivo se hace la prueba:** Aceptación, smoke (humo), sanity (sanidad)
  - **Con qué conocimientos se diseñan:** Caja blanca frente a Caja negra

Cómo se ejecutan

code URJC

- **Pruebas manuales**
  - La prueba que es **realizada por una persona** interactuando con el SUT.
  - El usuario puede seguir algún **orden** o **guión** o con pruebas **exploratorias**.
  - Pruebas exploratorias:
    - No tienen un guión específico.
    - El probador "explora" el sistema, con las hipótesis acerca de cómo debe comportarse en base a lo que en la aplicación ya se ha hecho y luego prueba esas hipótesis para ver si se sostienen.
    - Si bien no existe un plan rígido, las pruebas exploratorias son una actividad disciplinada con la que es más probable encontrar errores reales que con las pruebas de forma rígida con un guión.

Herramientas

- **Herramientas para gestionar pruebas**
  - Aplicación web que registra el plan de pruebas y los resultados de las ejecuciones tanto **manuales** como **automáticas**

http://www.softwaretestinghelp.com/15-best-test-management-tools-for-software-testers/

Cómo se ejecutan

- **Pruebas automáticas**
  - Pruebas que para su ejecución requieren la ejecución de un **código de pruebas** (código escrito específicamente para probar el SUT).
  - Es posible **verificar con mayor eficacia y eficiencia** el comportamiento del software porque se realiza de forma automática
  - Dependiendo de lo que haya que probar, **automatizar una prueba puede ser muy costoso**: Tecnologías de interfaz de usuario complejas de probar de forma automática.

Herramientas

- Frameworks y librerías para el **ciclo de vida y las verificaciones**

JUnit 5

Hamcrest

Jasmine

Jest

KARMA

mockito

Herramientas

- Librerías para **interactuar con el SUT**

REST-assured

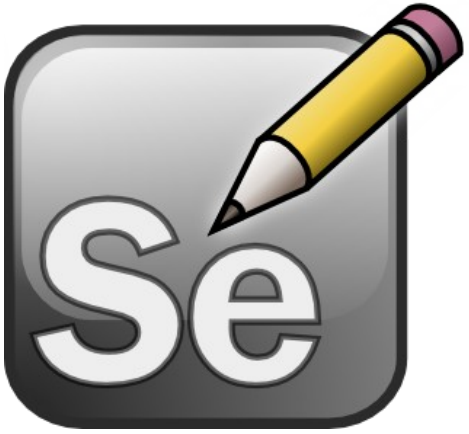
Protractor

end to end testing for AngularJS

Robotium

Herramientas

- Herramientas de grabación



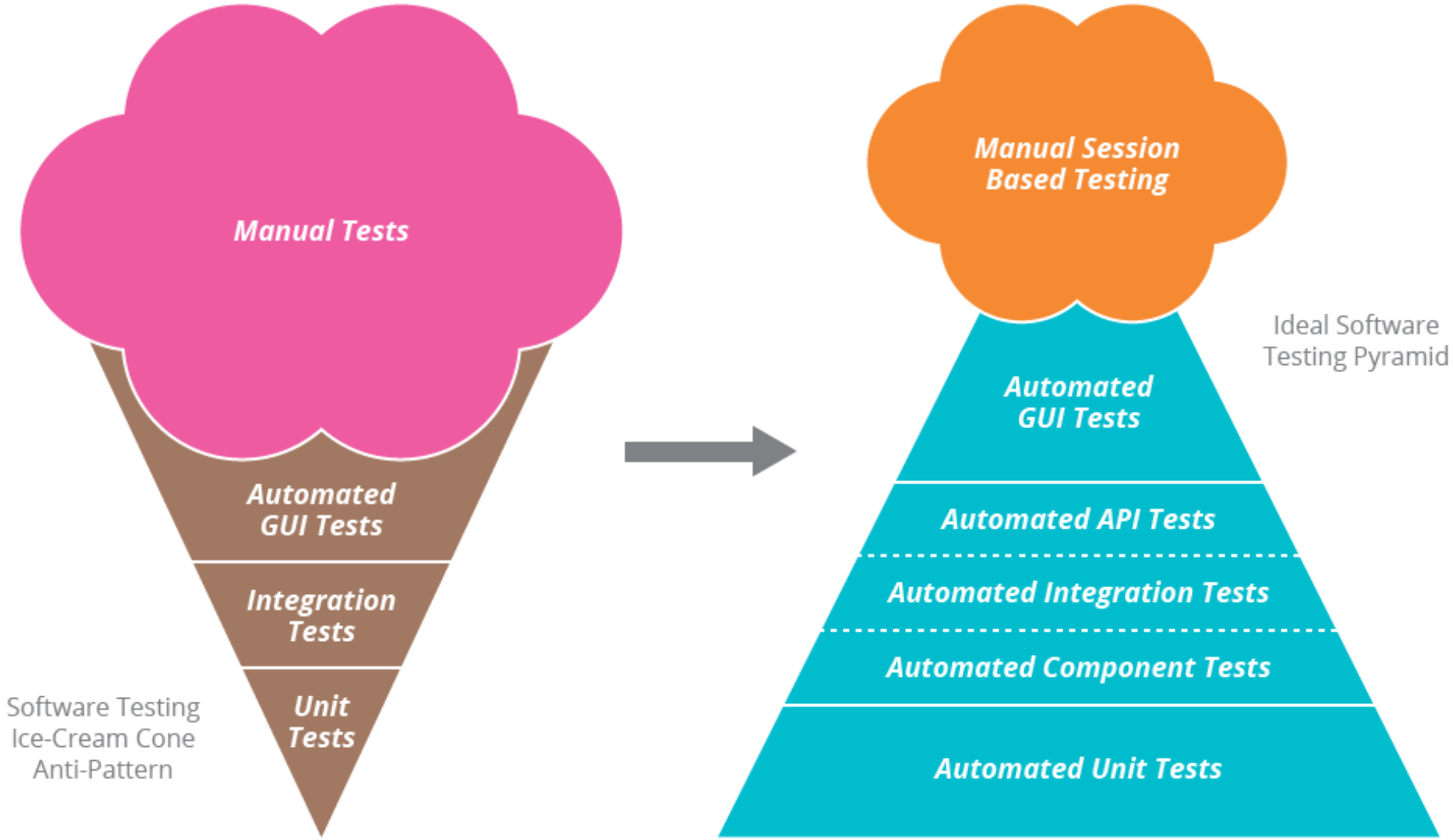
Selenium IDE

TestComplete

Cómo se ejecutan

- **Pruebas manuales vs automáticas**
  - **Hay pruebas que pueden realizarse de cualquier forma:**
    - Las pruebas de sistema, pruebas de integración cuando se usan protocolos de red para comunicación (API REST, Consultas a la BBDD...)
    - Es preferible que se automaticen para reducir el coste)
  - **Otras pruebas sólo tienen sentido de forma automática:** Pruebas unitarias, pruebas de integración, pruebas de carga, pruebas de estrés...
  - **Otras sólo tienen sentido de forma manual:** Pruebas de usabilidad, pruebas de sistema exploratorias...

Cómo se ejecutan



https://www.thoughtworks.com/insights/blog/architecting-continuous-delivery

Tipos de pruebas

- Existen muchas formas de clasificar las pruebas:
  - **Qué características prueban:** Funcionales o no funcionales
  - **Qué es el SUT:** Unidad, componente, integración, sistema
  - **Cómo se ejecutan:** Manuales o automáticas
  - **Con qué objetivo se hace la prueba:** Aceptación, smoke (humo), sanity (sanidad)
  - **Con qué conocimientos se diseñan:** Caja blanca frente a Caja negra

Con qué objetivo se hace la prueba

code URJC

- **Pruebas de aceptación:**
  - Para el ISTQB (*International Software Testing Qualifications Board*):
    - **Pruebas de sistema** que permiten validar si el sistema cumple con los objetivos para los que fue diseñado.
    - Se **ejecutan manualmente por usuarios del sistema**
    - Por ejemplo **versiones beta** que prueban los usuarios
  - Para la comunidad agile
    - Pruebas que definen las **reglas de negocio** que tiene que ofrecer la aplicación
    - Se ejecutan de forma **automática**
    - Pueden ser de **sistema**, de **integración** o incluso **unitarias**

https://en.wikipedia.org/wiki/Acceptance_testing

Con qué objetivo se hace la prueba

code URJC

- **Prueba de humo (smoke test):**
  - Prueba de sistema que tiene como objetivo saber si el sistema está desplegado. Es previa a la ejecución de pruebas más exhaustivas. Para una web, saber si atiende peticiones http. Usadas después de un despliegue.
- **Pruebas de sanidad (sanity check):**
  - Prueba las funcionalidades básicas del sistema. Usadas después de un

Tipos de pruebas

- Existen muchas formas de clasificar las pruebas:
  - **Qué características prueban:** Funcionales o no funcionales
  - **Qué es el SUT:** Unidad, componente, integración, sistema
  - **Cómo se ejecutan:** Manuales o automáticas
  - **Con qué objetivo se hace la prueba:** Aceptación, smoke (humo), sanity (sanidad)
  - **Con qué conocimientos se diseñan:** Caja blanca frente a Caja negra

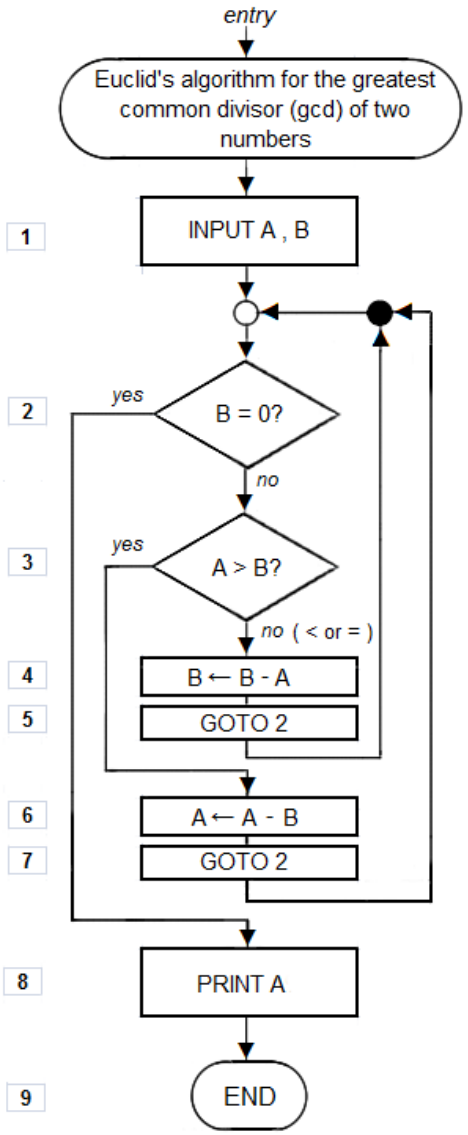
Con qué conocimientos se diseñan

code URJC

- **Caja Negra (Black-box):**
  - Sólo se tiene en cuenta la descripción de la **funcionalidad observable del SUT.**
  - Verifican si se obtiene la **salida deseada** a una **entrada dada**.
  - Se utiliza generalmente en pruebas de **sistema e integración**
  - Idealmente por **equipos no involucrados en el desarrollo** (para no estar contaminados por detalles de implementación o asunciones en la interpretación de requisitos)

Con qué conocimientos se diseñan

- **Caja blanca (White-box):**
  - Se tiene en cuenta cómo está construido el **SUT internamente**
  - Permite que todo el **código interno sea ejercitado** durante las pruebas. Todos los caminos independientes son ejecutados
  - Tienen que realizarse por los **desarrolladores** que han implementado el SUT.



Introducción

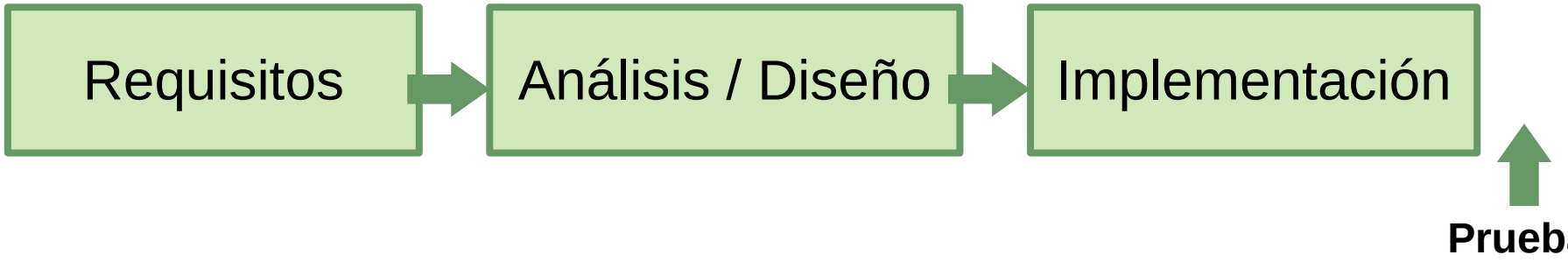
- Tipos de pruebas
- **Metodologías**
- JUnit 5
- Conclusiones

Metodologías

- En las metodologías clásicas la **fase de pruebas** siempre aparece después de la **implementación**.
- Existen **otras metodologías** en las que las pruebas tienen **diferentes ciclos de vida**.
- Fases de cualquier **ciclo de vida de desarrollo software:**

Desarrollo con Pruebas al Final

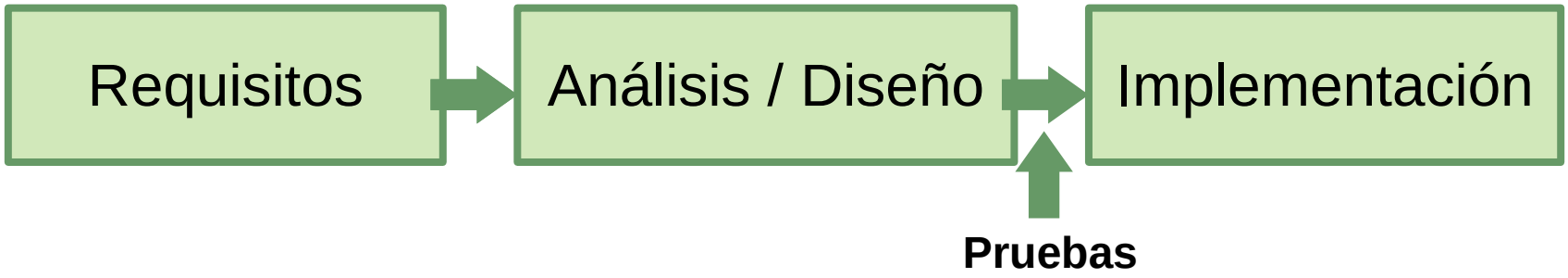
TLD (*Test Last Development*)



- El desarrollo de pruebas se realiza después de la implementación del SUT.
- Típicamente sólo se implementan **pruebas de sistema**
- Metodología: **Cascada**

Desarrollo con Pruebas al Principio

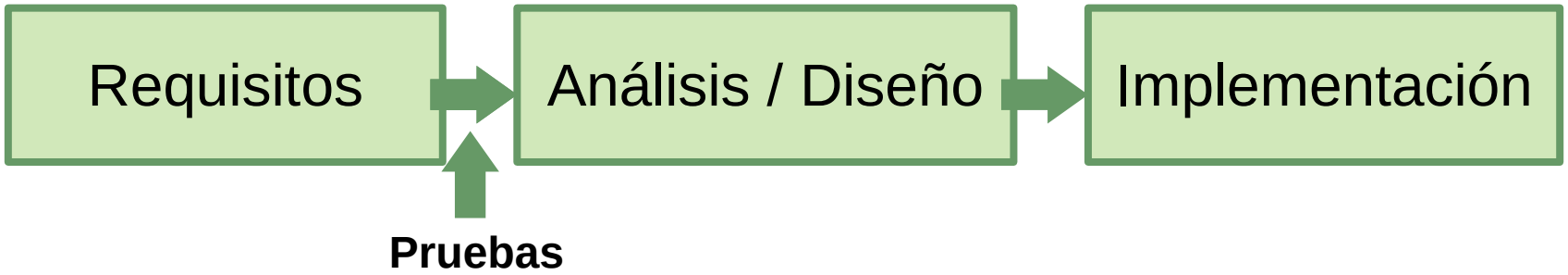
TFD (*Test First Development*)



- El desarrollo de pruebas se realiza antes de la fase de implementación, después del diseño y análisis
- Las responsabilidades de cada unidad se entienden antes de la implementación
- Metodología: **Proceso Unificado de Desarrollo**

Desarrollo dirigido por Pruebas

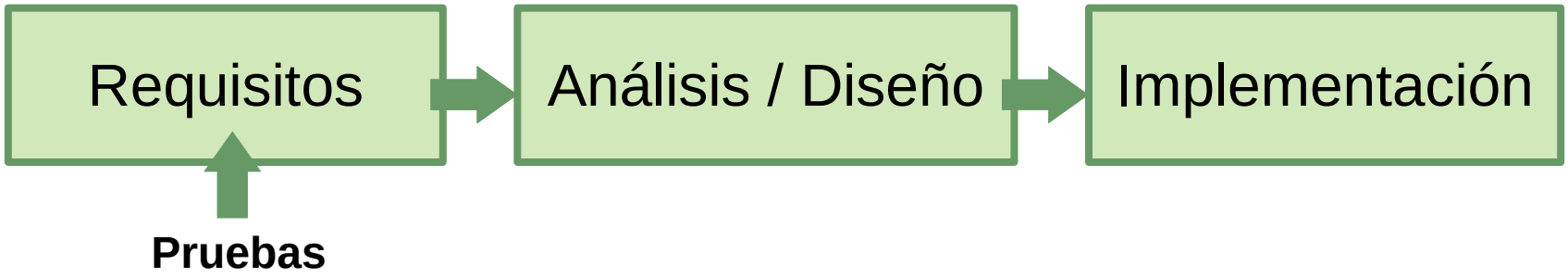
TDD (*Test Driven Development*)



- El desarrollo de pruebas se realiza junto con el análisis, diseño e implementación. **Las pruebas guían el desarrollo**
- El desarrollador no hace pruebas manuales mientras desarrolla, usa las **pruebas automáticas**
- Metodología: **Programación Extrema**

Desarrollo dirigido por Comportamiento

BDD (*Behavior Driven Development*)



- Los requisitos se definen con un formato parecido a lenguaje natural que es directamente ejecutable.
- **Los requisitos son pruebas automáticas**
- Metodología: **Ágiles**

Desarrollo dirigido por Comportamiento

BDD (*Behavior Driven Development*)

- Se usan herramientas específicas como **Cucumber y los requisitos se definen con el lenguaje Gherkin**

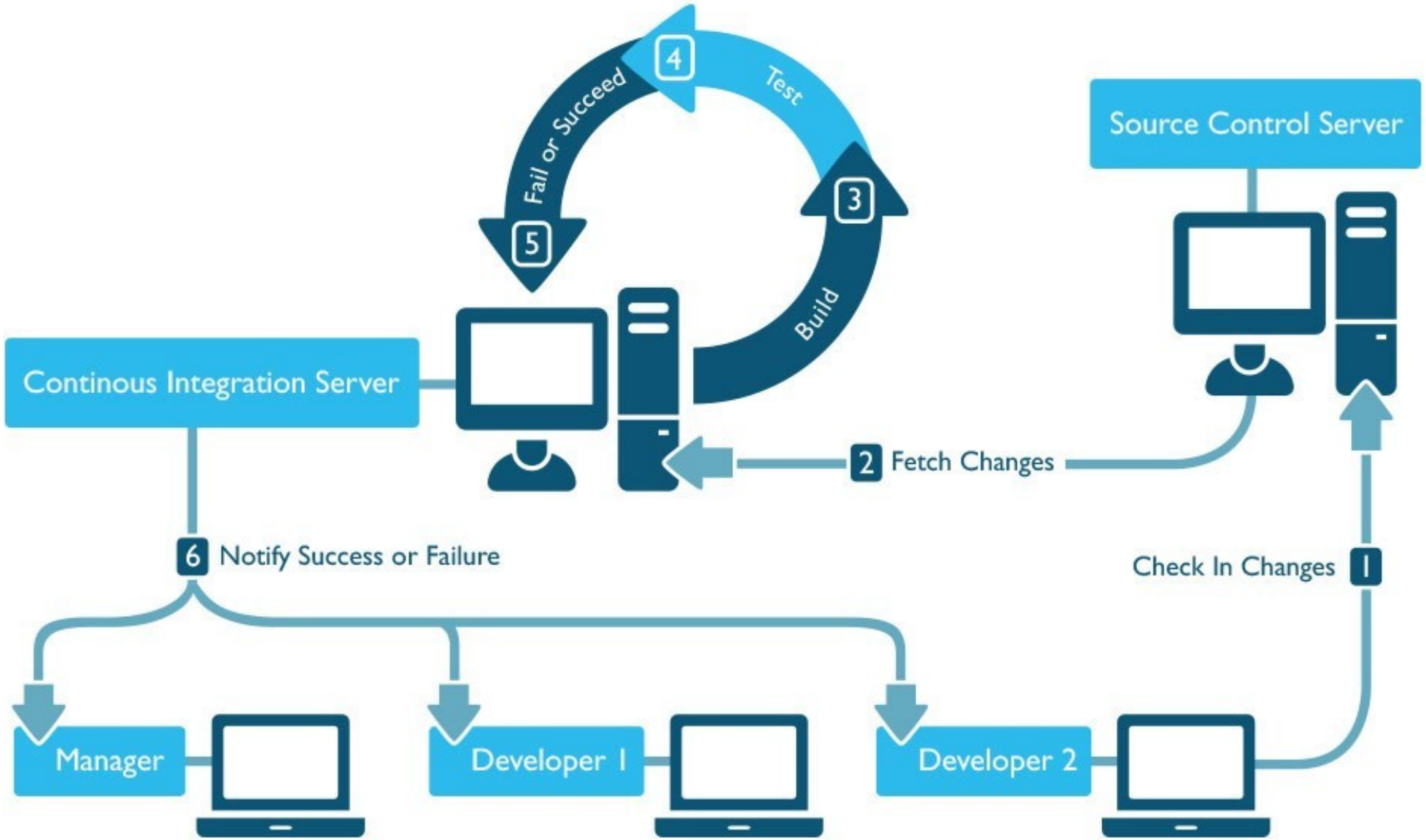
```
Feature: Login Page
    As a user I want to be able to login and out of the system.

    Background:
        Given I am at the login page

    Scenario: Login as a user
        When I login with user 'janedoe' and password 'password'
        Then I should see the home page
        And I can logout
```

Integración Continua

- Independientemente de la metodología empleada, se considera **buena práctica integrar de forma continua** el código desarrollado por los desarrolladores
- Para ello, se usan **repositorios compartidos de código** (con un sistema avanzado de gestión de versiones y conflictos)
- Cada vez que un desarrollador **sube código al repositorio** una herramienta de integración continua (CI) **ejecuta las pruebas automáticas** sobre el software

Integración Continua

code URJC



Integración Continua

- **Regresiones**
  - Disponer de **pruebas automáticas desde etapas tempranas** del desarrollo nos permite que el proceso de Integración Continua detecte las **regresiones**
  - Una **regresión** es la pérdida de una funcionalidad existente a consecuencia de un **nuevo código** añadido en el software
  - Por eso se usan cada vez más las metodologías que favorecen **tener tests cuanto antes:** BDD, TDD

Integración Continua

code URJC

- **Herramientas**

+

Repositorios de código

Servidores de Integración Continua

Calidad de las pruebas

code URJC

- Un **software se puede usar de formas infinitas**, pero no podemos diseñar y ejecutar infinitas pruebas
- **Cuántas pruebas** hay que implementar?
- Cuándo el software se puede considerar **suficientemente probado**?
- Existen **técnicas** para determinar la **calidad de las pruebas**

Calidad de las pruebas

- **Cobertura de código (*code coverage*)**
  - Es el % de código del SUT ejecutado cuando se ejecuta el conjunto de pruebas
  - Cobertura del 100%
    - No implica que el SUT no tenga defectos
    - Una misma línea de código puede ejecutarse correctamente con unos valores de variables e incorrectamente con otros.
    - A veces es muy complicado llegar a una cobertura del 100% porque hay código genérico que sólo se ejecuta en situaciones difíciles de probar
  - Se considera como valor aceptable una cobertura 70-80%

http://www.bullseye.com/minimum.html

Calidad de las pruebas

- **Cobertura de código (*code coverage*)**

JaCoCo

```
    public boolean addAll(int index, Collection c) {
        if(c.isEmpty()) {
            return false;
        } else if(_size == index || _size == 0) {
            return addAll(c);
        } else {
            Listable succ = getListableAt(index);
            Listable pred = (null == succ) ? null : succ.prev();
            Iterator it = c.iterator();
            while(it.hasNext()) {
                pred = insertListable(pred,succ,it.next());
            }
            return true;
        }
    }
```

1 of 2 branches missed.

Press 'F2' for focus

http://www.eclemma.org/

build passing coverage

Calidad de las pruebas

- **Mutaciones (*Mutation testing*)**
  - El código del SUT se modifica de forma automática (mutación) creando un nuevo SUT (mutante)
  - Se ejecutan los tests contra el SUT mutante

Si los tests fallan

- El mutante muere.
- Los tests son de calidad porque han detectado el cambio de comportamiento.

Si los tests no fallan

- El mutante vive.
- Los tests son mejorables porque no han sido capaces de detectar el cambio de comportamiento

Calidad de las pruebas

- **Mutaciones (*Mutation testing*)**

Mutation Testing para Java

Introducción

- Tipos de pruebas
- Metodologías
- **JUnit 5**
- Conclusiones

Casos de Test

- **JUnit 5**
  - La nueva versión de JUnit es una **actualización importante** sobre JUnit 4
  - Requiere **Java 8** porque se basa en sus funcionalidades (especialmente **expresiones lambda**)
  - Se ha refactorizado para que sea más fácil de **integrar en los IDEs** y en herramientas como **Maven**

http://junit.org/junit5/

http://blog.scottlogic.com/2017/10/10/junit-5.html

JUnit 5

code URJC

- Cambio de dependencia **Maven**

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-api</artifactId>
  <version>5.0.2</version>
  <scope>test</scope>
</dependency>
```

- **Jupiter** es el nuevo nombre que recibe la API Java para implementación de tests

- **Nuevos nombres**
  - Paquete

  - Clases y anotaciones

```
@Before          → @BeforeEach
@After           → @AfterEach
@BeforeClass     → @BeforeAll
@AfterClass      → @AfterAll
@Ignore          → @Disabled
Assert           → Assertions
```

- **Aserción de excepciones**
  - @Test ya no permite la propiedad “expected”
  - Se utiliza un enfoque similar a AssertJ
  - El código que genera la excepción se define en una expresión lambda para que se pueda capturar la excepción y verificar sus propiedades

- **Aserción de excepciones**

```
@Test(expected = IllegalArgException.class)
public void test() {
    try {
        testee.setAlertStatus(null);
        fail("Exception should have been thrown");
    } catch (IllegalArgException e) {
        assertEquals("Alert status cannot be set to null.", e.getMessage());
        throw e;
    }
}
```

```
@Test
void test() {
  IllegalArgException actual =
    assertThrows(IllegalArgException.class,()-> testee.setAlertStatus(null));
  assertEquals("Alert status cannot be set to null.", actual.getMessage());
}
```

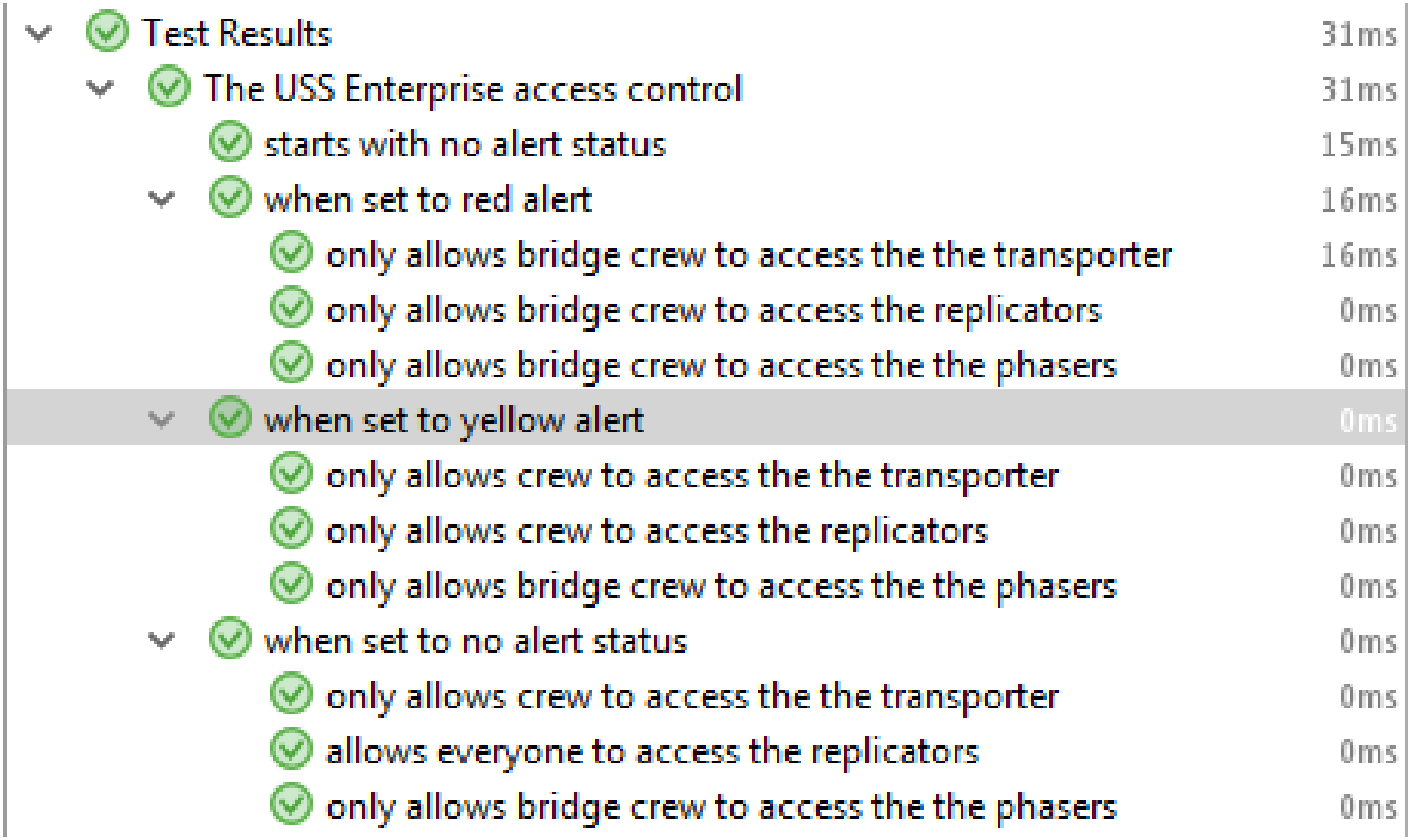
- **Tests más descriptivos**
  - DisplayName

```
@DisplayName("The USS Enterprise access control")
class EnterpriseAccessControlTest {

    ...

    @Test
    @DisplayName("starts with no alert status")
    void startsWithNoAlert() {
        assertEquals(Alert.NONE, testee.getAlertStatus());
    }
}
```

- **Tests más descriptivos**
  - Clases de tests anidadas

```
@DisplayName("The USS Enterprise access control")
class EnterpriseAccessControlTest {

    ...

    @Nested
    @DisplayName("when set to yellow alert")
    class whenYellowAlert {

        @Test
        @DisplayName("only allows crew to access replicators")
        void canAccessReplicators() {
            ...
        }
    }
}
```

- **Tests más descriptivos**



- **Tests parametrizados**
  - Una forma mucho más concisa de definir tests parametrizados
  - El método del test se anota con **@ParameterizedTest** en vez de @Test
  - Con una anotación se define los parámetros:
    - @ValueSource: Valores literales
    - @EnumSource: Valores de un enumerado

http://junit.org/junit5/docs/current/user-guide/#writing-tests-parameterized-tests

- **Tests parametrizados**

```
@ParameterizedTest
@ValueSource(strings = { "racecar", "radar", "able was I ere I saw elba" })
void palindromes(String candidate) {
    assertTrue(isPalindrome(candidate));
}
```

```
@ParameterizedTest
@EnumSource(TimeUnit.class)
void testWithEnumSource(TimeUnit timeUnit) {
    assertNotNull(timeUnit);
}
```

```
@ParameterizedTest
@MethodSource("stringProvider")
void testWithSimpleMethodSource(String argument) {
    assertNotNull(argument);
}
static Stream<String> stringProvider() {
    return Stream.of("foo", "bar");
}
```

- **Otras funcionalidades**
  - **Tests dinámicos** que se crean durante la ejecución. Con JUnit 4 los tests son métodos de clases
  - Aserciones con **timeouts**
  - **Múltiples extesiones** para la misma Test Class. Con JUnit 4 sólo se puede usar una extensión a la vez con **@RunWith**.
  - Filtrar en ejecución qué tests se ejecutan

Introducción

- Tipos de pruebas
- Metodologías
- JUnit 5
- **Conclusiones**

Conclusiones

- Las **pruebas automáticas** son una de las técnicas más usadas para **crear código con pocos defectos:**
  - Junto con la **integración continua se verifica** de forma continua que el software se **comporta como se espera**
  - **Evitan regresiones** durante el ciclo de desarrollo
  - Ayudan en el **desarrollo** y a la **comunicación con negocio** (TDD y BDD)
- Las **pruebas manuales** tienen su utilidad en pruebas de sistema **exploratorias** y en pruebas de **aceptación**

Conclusiones

- Implementar pruebas automáticas **no es sencillo**
  - Existen muchos **tipos de pruebas** (algunas más complejas de implementar que otras)
  - Existen **muchas librerías y herramientas de apoyo** para la implementación y ejecución de todos tipos de pruebas (síntoma de que no es sencillo)
  - Cada **lenguaje de programación / plataforma** tiene sus propias herramientas y librerías

Conclusiones

- Existen formas de medir la **calidad de las pruebas** automáticas
  - Cobertura del código (*Code coverage*)
  - Pruebas de mutación (*Mutation testing*)
- **Coste** de las pruebas automáticas
  - Implementar pruebas automáticas tiene asociado un coste de implementación y mantenimiento.
  - Hay que llegar a un compromiso de valor aportado (defectos que no llegan a producción) frente a coste.

2.2 – Pruebas de Servicios de Internet

Tema 1 – Introducción a Pruebas de Software

Universidad Rey Juan Carlos

Micael Gallego
micael.gallego@urjc.es
@micael_gallego

Francisco Gortázar
francisco.gortazar@urjc.es
@fgortazar

code URJC