



TDD: TESTING Y CALIDAD EN EL DESARROLLO SOFTWARE



ARTESANIA DEL SOFTWARE



¡HOLA!

Soy Rubén Valero García

Desarrollador de CRC it

rvalero@crcit.es



¿QUIÉN SOMOS?

CRC Information Technologies es una compañía española que desarrolla su actividad en el sector servicios, dentro del ámbito de las tecnologías de la información.

Desde su constitución, CRC Information Technologies ha tenido la satisfacción de poder desarrollar su trabajo para más de 25 empresas e instituciones, y nos alegra saber que uno de los aspectos que siempre ha destacado es el enorme valor, tanto técnico, como humano, del equipo de profesionales con los que cuenta.

CRC Information Technologies, con una plantilla de más de 50 empleados, en su gran mayoría ingenieros superiores, ofrece al mercado la experiencia acumulada a través de la realización de más de 1.000 proyectos tecnológicos.

<http://www.crcit.es/>

FORMACIÓN CRC IT

Los programas de formación que ofrece CRC Information Technologies pretenden cubrir las necesidades de sus clientes de modo global, desde la formación básica en aspectos informáticos hasta cursos especializados en tecnologías específicas.

CRC Information Technologies cuenta con profesionales con una dilatada experiencia en la utilización y formación en estas tecnologías. Para más información:

<http://www.crcit.es/formacion/>





BLOQUES DEL CURSO

1. Introducción

- Informe CHAOS
- Código limpio



2. Test y código limpio

- Responsabilidades
- Código limpio



3. JUnit

- xUnit
- API



4. Pruebas Software: Teoría

- Historia
- Tipología
- Escritura



TDD

- Descripción
- Algoritmo TDD
- BDD, DDD, ADD, TDD



DOBLES DE PRUEBA

- Tipos de dobles
- Usos
- Mockito



TEST DE CLIENTE Y DE SISTEMA

- Cuándo y cómo probar el cliente
- Jasmine
- Selenium



IC Y HERRAMIENTAS DE CALIDAD Y VALIDACIÓN

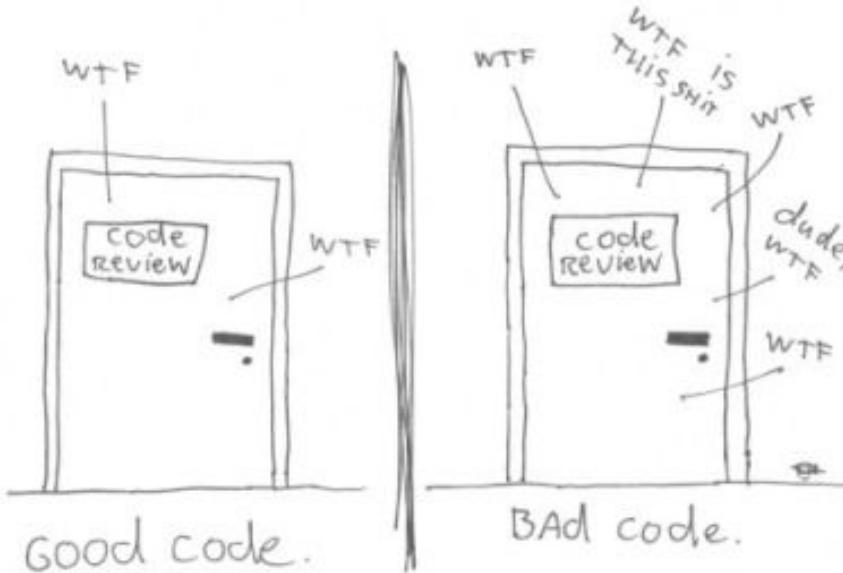
- Definiciones
- Empezando..
- Jenkins
- JMeter

INTRODUCCIÓN

1.1 ¿Que es un test software?



The ONLY VALID MEASUREMENT
OF CODE QUALITY: WTFs/MINUTE



(c) 2008 Focus Shift

“

Según la Wikipedia:

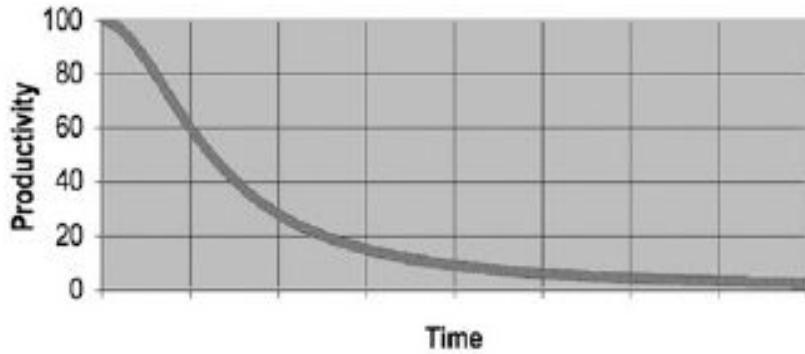
Las pruebas de software (en inglés software testing) son las investigaciones empíricas y técnicas cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto a la parte interesada o stakeholder. Es una actividad más en el proceso de control de calidad.



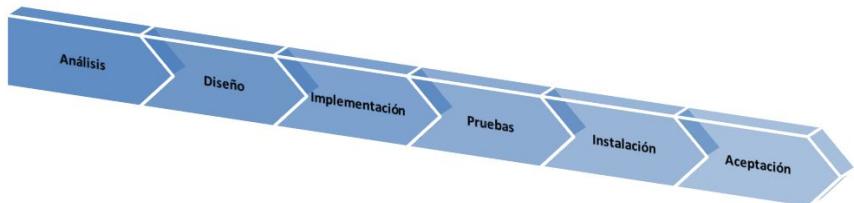
INTRODUCCIÓN

1.2 ¿Por qué probar?



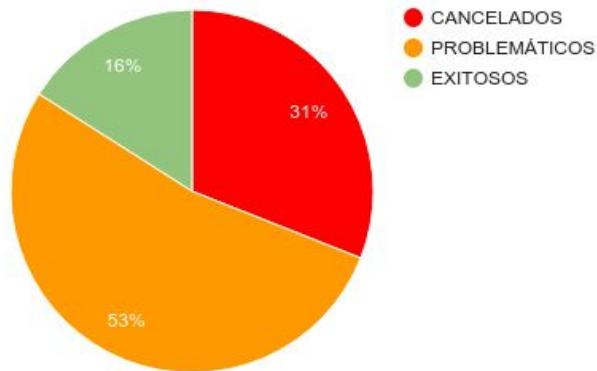


El coste total de un desastre

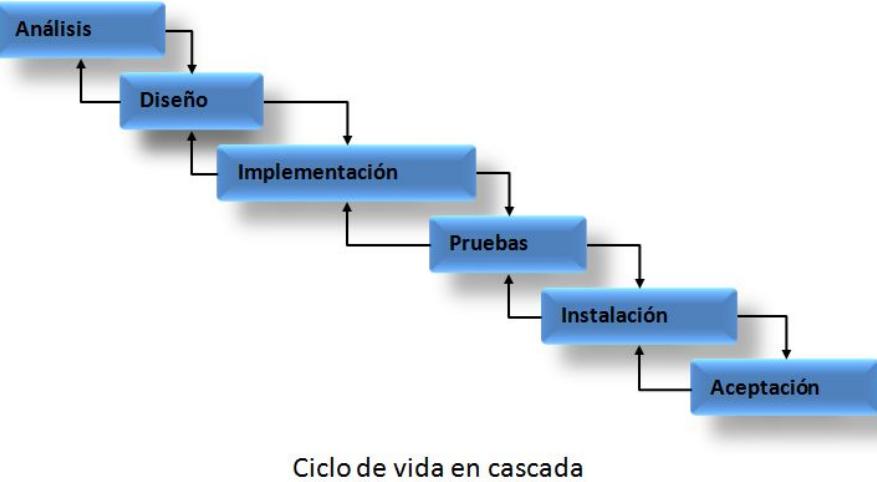


Ciclo de vida lineal

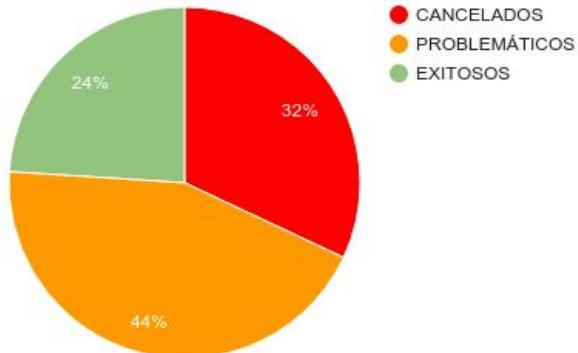
1994



*Porcentaje resultados desarrollos
software 1994*



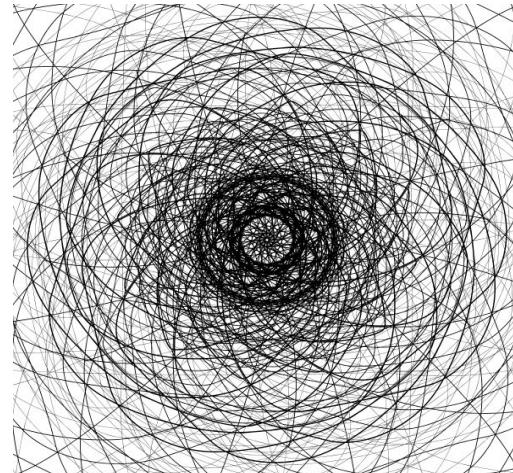
2009



Informe CHAOS

*Porcentaje resultados desarrollos
software 2009*

- Escasa participación de los usuarios.
- Requerimientos y especificaciones incompletas.
- Cambios frecuentes en los requerimientos y especificaciones.
- Falta de soporte ejecutivo.
- Incompetencia tecnológica.
- Falta de recursos.
- Expectativas no realistas.
- Objetivos poco claros.
- Cronogramas irreales.
- Nuevas tecnologías.





EL CÓDIGO LIMPIO

Como desarrolladores nuestra responsabilidad es la
CALIDAD del código:

<http://agilemanifesto.org/>



PRUEBAS Y CÓDIGO LIMPIO

Test como motor de código de calidad

2.1 ¿Que es un test software?





¿Qué constante de diseño se aplica a todos los desarrollos, de todas las tecnologías, en todas las épocas y todos los planetas?



SEGÚN T. FREEMAN Y
E. ROBSON*..

Cambios



* Autores de la serie de libros Head First

- Generar documentación dinámicamente.
- Documentar el código (me dicen cómo y para qué funciona).
- Son pieza fundamental para la **REFACTORIZACIÓN** (**patrones**, mejoras en el código).
- Son la única manera realista de acometer **CAMBIOS** ágilmente.
- ¡Son la única manera inteligente de acometer **CAMBIOS** ágilmente!.
- ¡¡Son la única manera de acometer **CAMBIOS** ágilmente!!.
- Prueban que el correcto funcionamiento del código.
- Mediante un sistema de IC pueden ser la interfaz para una comunicación realista del desarrollo de un proyecto.
- **LOS TEST SIRVEN PARA CREAR CÓDIGO LIMPIO. CALIDAD DE SOFTWARE.**



REGLA DEL BOY SCOUT

Dejar el campamento más limpio de lo que te lo
has encontrado

(VS

¡Voy a tocar lo mínimo posible para no prepararla!)



PRUEBAS Y CÓDIGO LIMPIO

2.2 Código limpio



- Naming:
 - uso de nombres que revelen las intenciones
 - evitar la desinformación
 - que se puedan pronunciar
 - que se puedan buscar fácilmente
 - evitar “ *utils ”, “ *Manager ”, “ *Info ”,..
- Duplicidad
- Complejidad: Equilibrio entre duplicidad y complejidad, mejor tener duplicidad a una abstracción inadecuada.
- Ausencia de principios de diseño => Principio de mínima sorpresa (WTFs/minute).



“

*Debes poner el mismo cuidado nombrando a
una variable que nombrando a tu
primogénito (Robert C Martin)*

Las cuatro reglas de diseño según Kent Beck:

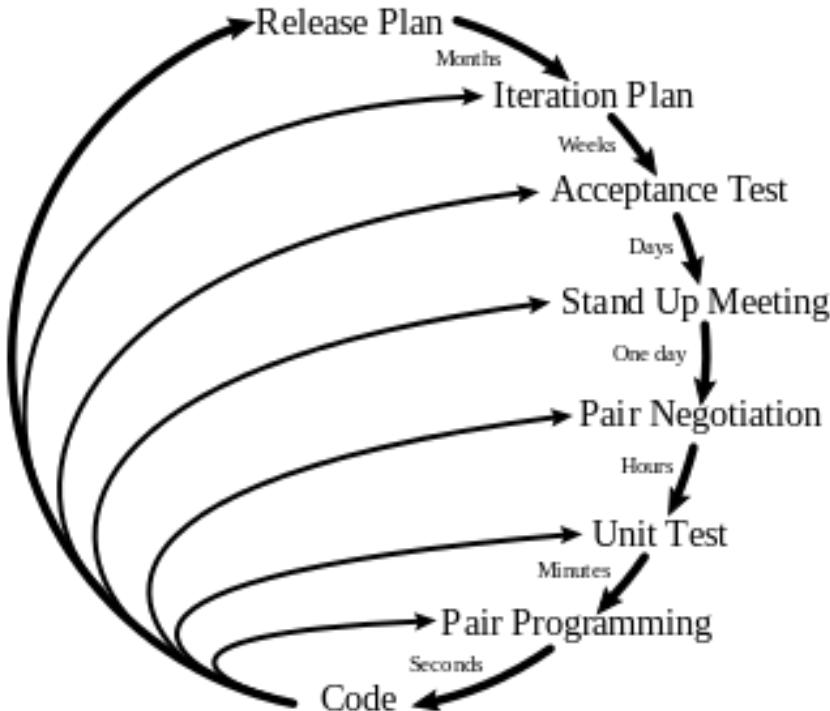
- **Ejecuta todas las pruebas.**
 - La creación de pruebas conduce a obtener mejores diseños, ya que afecta al cumplimiento por parte de nuestro sistema de los principales objetivos de la programación de bajo acoplamiento y elevada cohesión.
- **No contiene duplicados.**
- **Expresa la intención del programador.**
- **Minimiza el número de clases y métodos.**





SOLID

Planning/Feedback Loops



- **Feedback.**
- **Comunicación.**
- **Respeto.**
- **Simplicidad**

http://en.wikipedia.org/wiki/Extreme_programming
<http://www.extremeprogramming.org/>

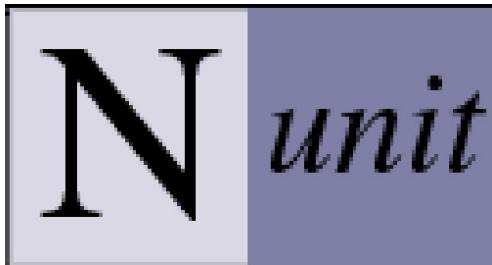
3

xUNIT

3.1 xUnit



JUnit



PL/Unit - Test Driven
Development for
Oracle

SUNIT

xUnit.net

http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks

3

JUNIT 3.2 API



ASSERTIONS

```
@Test
public void testAssertArrayEquals() {
    byte[] expected = "trial".getBytes();
    byte[] actual = "trial".getBytes();
    assertArrayEquals("failure - byte arrays not same", expected, actual);
}

@Test
public void testAssertEquals() {
    assertEquals("failure - strings are not equal", "text", "text");
}

@Test
public void testAssertFalse() {
    assertFalse("failure - should be false", false);
}

@Test
public void testAssertNotNull() {
    assertNotNull("should not be null", new Object());
}

@Test
public void testAssertNotSame() {
    assertNotSame("should not be same Object", new Object(), new Object());
}

@Test
public void testAssertNull() {
    assertNull("should be null", null);
}

@Test
public void testAssertSame() {
    Integer aNumber = Integer.valueOf(768);
    assertEquals("should be same", aNumber, aNumber);
}
```

AGREGACIÓN DE TEST EN SUITES

```
@RunWith(Suite.class)
@Suite.SuiteClasses({
    TestFeatureLogin.class,
    TestFeatureLogout.class,
    TestFeatureNavigate.class,
    TestFeatureUpdate.class
})
public class FeatureTestSuite {
    ...
}

private class TestFeatureLogin {
}

private class TestFeatureLogout {
}

private class TestFeatureNavigate {
}

private class TestFeatureUpdate {
}
```

```
@Test(expected= IndexOutOfBoundsException.class)
public void empty() {
    new ArrayList<Object>().get(0);
}
```

assertThat([value], [matcher statement]);

Expresividad y legibilidad

assert “x is 3” en lugar de assert “equals 3 x”

Sujeto, verbo y predicado



El método assertThat coje un objeto y la **implementación** de un matcher

ASSERT vs ASSERTTHAT

```
String responseString = "c";  
  
@Test  
public void assertTrueTest() {  
    assertTrue(responseString.contains("color") || responseString.contains("colour"));  
}  
  
[  ]: 2 of 2 Failed: 2 (in 0,086 s)  
  
junit.framework.AssertionFailedError  
+  at practica3.parser.sol.junit.assertTrueTest(junit.java:19) <23 internal calls>
```



Assert

```
@Test  
public void assertThatTest() {  
    assertThat(responseString, anyOf(containsString("color"), containsString("colour")));  
}  
  
[  ]: 2 of 2 Failed: 2 (in 0,089 s)  
  
/usr/lib/jvm/java-8-oracle/bin/java ...  
  
java.lang.AssertionError:  
Expected: (a string containing "color" or a string containing "colour")  
      but: was "Please choose a font!"  
<Click to see difference>
```



AssertThat

MATCHERS

	Matcher	Descripción
Core	any()	Coincidencia de algún elemento
	is()	Coincide si un objeto es igual (equals) a otro.
	describedAs()	Añade una descripción a un Matcher
Logical	allOf()	Para un array de matchers todos los objetos deben coincidir con el objeto objetivo.
	anyOf()	Igual que el anterior pero sólo requiere la coincidencia de algún elemento
	not()	Negación de un matcher anterior

MATCHERS

Matcher	Descripción
Object	<code>equalTo()</code> Igualdad de dos objetos dados
	<code>instanceOf()</code> Matcher de instaceOf
	<code>notNullValue()</code> Evalúa si un objeto es null o not null
	<code>nullValue()</code>
<code>sameInstance()</code>	Evalúa si un objeto es exactamente la misma instancia que otro

BEFORE & AFTER

```
File output;
@Before
public void createOutputFile() {
    output= new File(...);
}
@Test public void something() {
    ...
}
@After
public void deleteOutputFile() {
    output.delete();
}
```

BEFORECLASS & AFTERCLASS

```
static DatabaseConnection database;
@BeforeClass
public static void login() {
    database= ...;
}
@Test public void something() {
    ...
}
@Test public void somethingElse() {
    ...
}
@AfterClass
public static void logout() {
    database.logout();
}

private class DatabaseConnection {
    public void logout() {
```

PARAMETERIZED TEST I

```
0  public class JunitAdditionTest {  
1  
2      private int expected=3;  
3      private int first=1;  
4      private int second=2;  
5  
6      @Test  
7      public void sum() {  
8          Addition add = new Addition();  
9          System.out.println("Addition with parameters : " + first + " and " + second);  
0          assertEquals(expected, add.AddNumbers(first, second));  
1      }  
2  
3      class Addition {  
4          public int AddNumbers(int first, int second) { return first + second; }  
5      }  
6  
7      }  
8
```

Done: 1 of 1 (in 0,014 s)

/usr/lib/jvm/java-8-oracle/bin/java ...

PARAMETERIZED TEST II

```
private int expected;
private int first;
private int second;

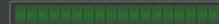
public JUnitAdditionParametrizedTest(int expectedResult, int firstNumber, int secondNumber) {
    this.expected = expectedResult;
    this.first = firstNumber;
    this.second = secondNumber;
}

@Parameterized.Parameters
public static Collection<Integer[]> addedNumbers() {
    return Arrays.asList(new Integer[][]{ {3, 1, 2}, {5, 2, 3}, {7, 3, 4}, {9, 4, 5}, });
}

@Test
public void sum() {
    Addition add = new Addition();
    System.out.println("Addition with parameters : " + first + " and " + second);
    assertEquals(expected, add.AddNumbers(first, second));
}

class Addition {
    public int AddNumbers(int first, int second) {
        return first + second;
    }
}
```

ie: 1 of 1 (in 0,014 s)



ASSUMPTIONS

```
    @Test public void filenameIncludesUsername() {  
        assumeThat(File.separatorChar, is('/'));  
        assertThat(new User("optimus").configFileName(), is("configfiles/optimus.cfg"));  
    }  
  
    private class User {  
        public User(String optimus) {  
        }  
  
        public Void configFileName() {  
        }  
    }  
}
```

RULE: GLOBAL TIMEOUT

```
public static String log;

@Rule
public TestRule globalTimeout = new Timeout(20);

@Test
public void testFastLoop() {
    for(int i=0; i<100; i++) {}
}

//==>Done: Process finished with exit code 0

@Test
public void testInfiniteLoop1() {
    log+= "ran1";
    for(;;) {}
}

//==>Failed: org.junit.runners.model.TestTimedOutException: test timed out after 20 milliseconds

@Test
public void testInfiniteLoop2() {
    log+= "ran2";
    for(;;) {}
}

//==>Failed: org.junit.runners.model.TestTimedOutException: test timed out after 20 milliseconds
```

RULE: TEMPORARY FOLDER

```
@Rule  
public TemporaryFolder folder = new TemporaryFolder();  
  
@Test  
public void testUsingTempFolder() throws IOException {  
    File createdFile = folder.newFile("myfile.txt");  
    File createdFolder = folder.newFolder("subfolder");  
    // ...  
}
```

RULE: EXTERNAL RESOURCE

```
Server myServer = new Server();

@Rule
public ExternalResource resource = new ExternalResource() {
    @Override
    protected void before() throws Throwable {
        myServer.connect();
    }

    @Override
    protected void after() {
        myServer.disconnect();
    }
};

@Test
public void testFoo() {
    new Client().run(myServer);
}
```

RULE: CUSTOM RULES

```
class MockRule implements TestRule {  
  
    private final Object target;  
  
    MockRule(Object target) {  
        this.target = target;  
    }  
  
    @Override  
    public Statement apply(final Statement statement, Description description) {  
        return new Statement() {  
            @Override  
            public void evaluate() throws Throwable {  
                MockitoAnnotations.initMocks(target);  
                statement.evaluate();  
            }  
        };  
    }  
}
```

3

JUNIT

3.3 xUnit



Facts are tests which are always true. They test invariant conditions.

```
using Xunit;

namespace MyFirstUnitTests
{
    public class Class1
    {
        [Fact]
        public void PassingTest()
        {
            Assert.Equal(4, Add(2, 2));
        }

        [Fact]
        public void FailingTest()
        {
            Assert.Equal(5, Add(2, 2));
        }

        int Add(int x, int y)
        {
            return x + y;
        }
    }
}
```

```
using Xunit;

namespace MyFirstUnitTests
{
    public class Class1
    {
        [Fact]
        public void PassingTest()
        {
            Assert.Equal(4, Add(2, 2));
        }

        [Fact]
        public void FailingTest()
        {
            Assert.Equal(5, Add(2, 2));
        }

        int Add(int x, int y)
        {
            return x + y;
        }
    }
}
```

Theories are tests which are only true for a particular set of data.

CONSTRUCTOR AND DISPOSE

```
public class StackTests : IDisposable
{
    Stack<int> stack;

    public StackTests()
    {
        stack = new Stack<int>();
    }

    public void Dispose()
    {
        stack.Dispose();
    }
}
```

```
[Fact]
public void WithNoItems_CountShouldReturnZero()
{
    var count = stack.Count;

    Assert.Equal(0, count);
}

[Fact]
public void AfterPushingItem_CountShouldReturnOne()
{
    stack.Push(42);

    var count = stack.Count;

    Assert.Equal(1, count);
}
```

Limpia el contexto para cada test

CLASS FIXTURES

```
public class DatabaseFixture : IDisposable
{
    public DatabaseFixture()
    {
        Db = new SqlConnection("MyConnectionString");

        // ... initialize data in the test database ...
    }

    public void Dispose()
    {
        // ... clean up test data from the database ...
    }

    public SqlConnection Db { get; private set; }
}

public class MyDatabaseTests : IClassFixture<DatabaseFixture>
{
    DatabaseFixture fixture;

    public MyDatabaseTests(DatabaseFixture fixture)
    {
        this.fixture = fixture;
    }

    // ... write tests, using fixture.Db to get access to the SQL Server ...
}
```

Creación de un contexto de test compartido para todos los test de una clase.

El contexto se limpia al terminar todos los test de la clase

COLLECTION FIXTURES

```
public class DatabaseFixture : IDisposable
{
    public DatabaseFixture()
    {
        Db = new SqlConnection("MyConnectionString");

        // ... initialize data in the test database ...
    }

    public void Dispose()
    {
        // ... clean up test data from the database ...
    }

    public SqlConnection Db { get; private set; }
}

[CollectionDefinition("Database collection")]
public class DatabaseCollection : ICollectionFixture<DatabaseFixture>
{
    // This class has no code, and is never created. Its purpose is simply
    // to be the place to apply [CollectionDefinition] and all the
    // ICollectionFixture<> interfaces.
}
```

```
[Collection("Database collection")]
public class DatabaseTestClass1
{
    DatabaseFixture fixture;

    public DatabaseTestClass1(DatabaseFixture fixture)
    {
        this.fixture = fixture;
    }
}

[Collection("Database collection")]
public class DatabaseTestClass2
{
    // ...
}
```

RUNNING TEST IN PARALLEL

```
[Collection("Our Test Collection #1")]
public class TestClass1
{
    [Fact]
    public void Test1()
    {
        Thread.Sleep(3000);
    }
}

[Collection("Our Test Collection #1")]
public class TestClass2
{
    [Fact]
    public void Test2()
    {
        Thread.Sleep(5000);
    }
}
```

PRUEBAS SOFTWARE: TEORÍA

4.1 Antes... Ahora?



HISTORIA PRUEBAS SOFTWARE





**El código está
incompleto sin test**



PRUEBAS SOFTWARE: TEORÍA

4.2 Tipos de test

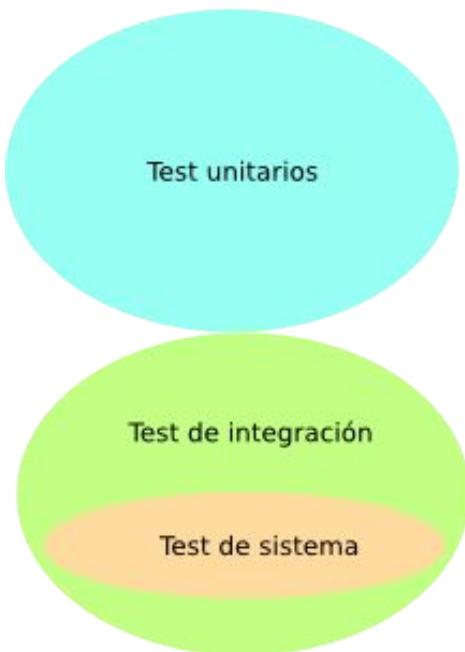


TIPOLOGÍA DE LAS PRUEBAS SOFTWARE



<http:// cwd.dhemery.com/2004/04/dimensions/>
http://en.wikipedia.org/wiki/Software_testing

Desarrolladores



Dueño del producto

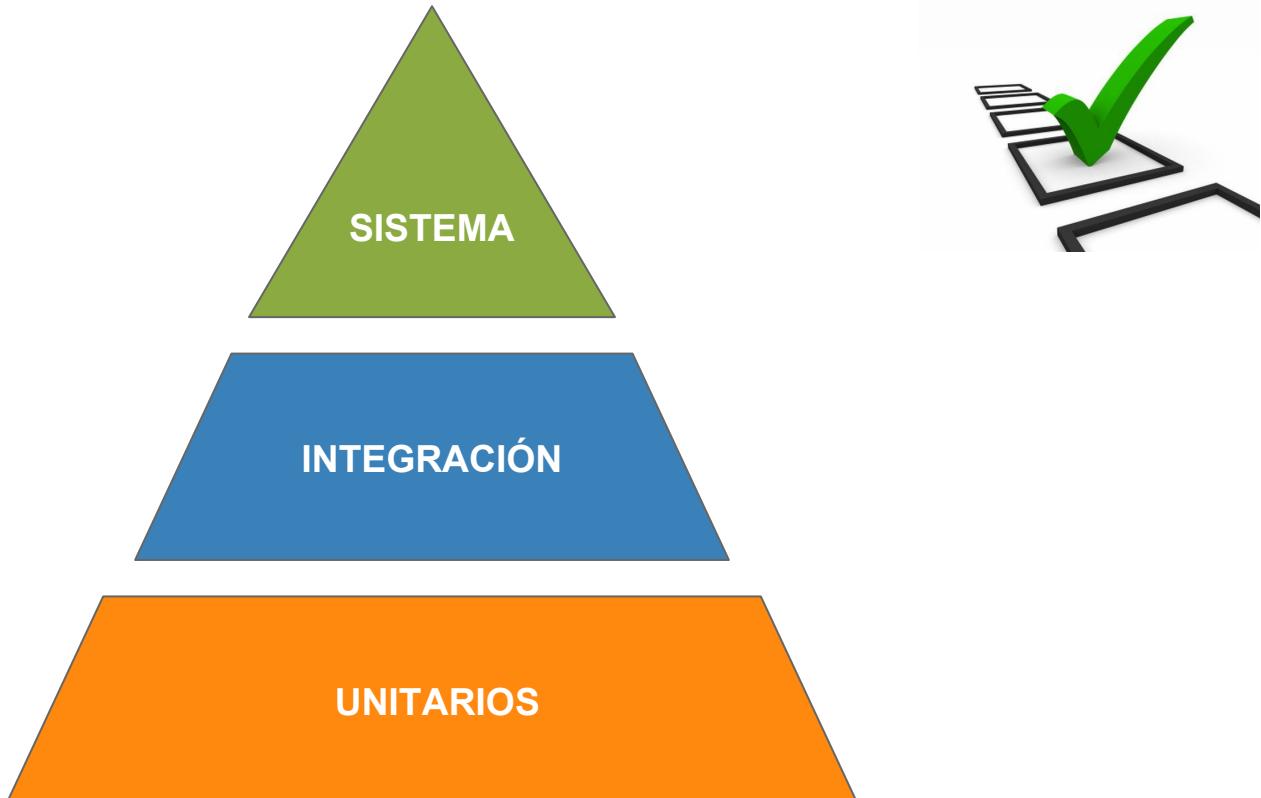


TEST DE ACEPTACIÓN
vs
TEST FUNCIONALES



“Para acceder a la solicitud que se quiera modificar se deberá introducir la identificación de la solicitud. Únicamente el solicitante, o el usuario con permisos de administrador podrán realizar esta modificación”

TEST UNITARIOS, DE INTEGRACIÓN Y SISTEMA





Los test de sistema son muy frágiles en el sentido de que cualquier cambio en cualquiera de las partes puede romperlos, y no tienen por qué revelar estrictamente el código concreto por el que se han roto.



SUT

Subject Under Test; El código que estamos probando



**Los test unitarios deberían atacar y
cubrir todos los aspectos del
comportamiento indivisible de
negocio.**

PRUEBAS SOFTWARE: TEORÍA

4.3 Escritura de los test





¿Qué constante de diseño se aplicaba a todos los desarrollos, de todas las tecnologías, en todas las épocas y todos los planetas?



SEGÚN T.
FREEMAN Y E.
ROBSON*..



Change



* Autores de la serie de libros Head First

“

*Si las pruebas no están limpias y se mantienen, las perderás. Y al perderlas, perderás la pieza más importante para articular **cambios** en el código de producción. Sin pruebas, cada cambio es un posible error. Independientemente de la arquitectura, el diseño,.. sin pruebas no podremos estar seguros de no “romper” alguna funcionalidad del código*





**El código de prueba es tan
importante como el código de
producción**

CARACTERÍSTICAS MÁS IMPORTANTES DE LOS TEST



* Clean Code Robert C Martín

REFACTORIZACIÓN DE PRUEBAS

```
@Test  
public void turnOnLoTempAlarmAtThreshold() {  
    hw.setTemp(WAY_TOO_COLD);  
    controller.tic();  
    Assert.assertTrue(hw.heaterState());  
    Assert.assertTrue(hw.blowerState());  
    Assert.assertFalse(hw.coolerState());  
    Assert.assertFalse(hw.hiTempAlarm());  
    Assert.assertTrue(hw.loTempAlarm());  
}  
|
```



```
@Test  
public void turnOnLoTempAlarmAtThreshold() {  
    wayTooCold();  
    assertEquals("HBchL", hw.getState());  
}  
  
public String getState() {  
    String state = "";  
    state += heater ? "H":"h";  
    [...]  
    return state;  
}
```





F.I.R.S.T.

- Fast
- Independent
- Repeatable
- Shelf-validating
- Timely



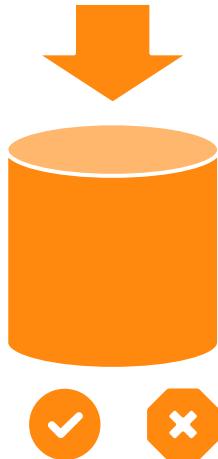
A.A.A

- Arrange
- Act
- Assert

```
@Test  
public void firstLetterUpperCase() {  
    String name = "pablo lópez";  
    NameNormalizer normalizer = new NameNormalizer();  
    String result = normalizer.firstLetterUpperCase(name);  
    Assert.assertEquals("Pablo López", result);  
}  
  
@Test  
public void surnameFirst() {  
    String name = "enrique arnáz";  
    NameNormalizer normalizer = new NameNormalizer();  
    String result = normalizer.surnameFirst(name);  
    Assert.assertEquals("arnaz, enrique", result)  
}
```



```
NameNormalizer normalizer;  
  
@Before  
public void setUp() {  
    normalizer = new NameNormalizer();  
}  
  
@Test  
public void firstLetterUpperCase() {  
    String name = "pablo lópez";  
  
    String result = normalizer.firstLetterUpperCase(name);  
  
    Assert.assertEquals("Pablo López", result);  
}  
  
@Test  
public void surnameFirst() {  
    String name = "enrique arnáz";  
  
    String result = normalizer.surnameFirst(name);  
  
    Assert.assertEquals("arnaz, enrique", result)
```

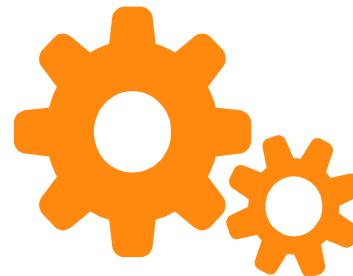


ESTADO

Test de entrada-Salida

INTERACCIÓN

Test de comportamiento entre colaboradores





Valida las excepciones

Al usar excepción genérica, estas enmascarando
fallos del SUT

- **Pruebas insuficientes:** Las pruebas han de probar todo lo que puede fallar.Y han de documentar los fallos
- **Usar una herramienta de cobertura**
- **No ignorar pruebas triviales.** Son fáciles de escribir y su valor documental aporta más que lo que cuesta crearlas
- **Una prueba ignorada es una pregunta sin respuesta**
- **Probar condiciones límite.** Solo acertar en la parte central de un algoritmo y errar en sus límites
- **Probar de forma exhaustiva los errores.** Al detectar un error, haga una prueba que falle, y compruebe que los cambios en producción que la hacen pasar no rompen otras pruebas
- **Las pruebas deben ser rápidas**
- **Las pruebas lentas deben correr en un CI periódicamente**

PRUEBAS SOFTWARE: TEORÍA

4.3 Pruebas de aceptación con JBehaviour



HISTORIAS

```
EjemploCalculadora.java x  Math.story x 
Narrative:  
In order to probar una calculadora  
  
Scenario: 2 x 2  
  
Given una variable x con valor 2  
When multiplico x por 2  
Then x deberia ser igual a 4  
  
Scenario: 3 x 3  
  
Given una variable x con valor 3  
When multiplico x por 3  
Then x deberia ser igual a 9|
```

CARGA DE HISTORIAS

```
public class SimpleJBehave extends JUnitStories {  
  
    private Configuration configuration;  
  
    public SimpleJBehave() {  
        super();  
        configuration = new Configuration() {  
            // configuration.doDryRun(false); "no dry run" is implicit by using  
            // default StoryControls  
  
            // configuration.useDefaultStoryReporter(new ConsoleOutput());  
            // deprecated -- rather use StoryReportBuilder  
  
            configuration.useFailureStrategy(new RethrowingFailure());  
            configuration.useKeywords(new LocalizedKeywords(Locale.ENGLISH));  
            configuration.usePathCalculator(new AbsolutePathCalculator());  
            configuration.useParameterControls(new ParameterControls());  
            configuration.useParameterConverters(new ParameterConverters());  
            configuration.useParanamer(new NullParanamer());  
            configuration.usePendingStepStrategy(new PassingUponPendingStep());  
            configuration.useStepCollector(new MarkUnmatchedStepsAsPending());  
            configuration.useStepdocReporter(new PrintStreamStepdocReporter());  
            configuration.useStepFinder(new StepFinder());  
            configuration.useStepMonitor(new SilentStepMonitor());  
            configuration  
                .useStepPatternParser(new RegexPrefixCapturingPatternParser())  
            configuration.useStoryControls(new StoryControls());  
            configuration.useStoryLoader(new LoadFromClasspath());  
            configuration.useStoryParser(new RegexStoryParser(configuration  
                    .getConfiguration().getStoryPathResolver().getStoryPath());  
        };  
    }  
}
```

```
        configuration.useStoryLoader(new LoadFromClasspath());  
        configuration.useStoryParser(new RegexStoryParser(configuration  
            .getConfiguration().getStoryPathResolver().getStoryPath()));  
        configuration.useStoryPathResolver(new UnderscoredCamelCaseResolver());  
        configuration.useStoryReporterBuilder(new StoryReporterBuilder());  
        configuration.useViewGenerator(new FreemarkerViewGenerator());  
  
        EmbedderControls embedderControls = configuredEmbedder()  
            .embedderControls();  
        embedderControls.doBatch(false);  
        embedderControls.doGenerateViewAfterStories(true);  
        embedderControls.doIgnoreFailureInStories(false);  
        embedderControls.doIgnoreFailureInView(false);  
        embedderControls.doSkip(false);  
        embedderControls.doVerboseFailures(false);  
        embedderControls.doVerboseFiltering(false);  
        embedderControls.useStoryTimeoutInSecs(300);  
        embedderControls.useThreads(1);  
    }  
  
    @Override  
    public Configuration configuration() { return configuration; }  
  
    @Override  
    public InjectableStepsFactory stepsFactory() {  
        return new InstanceStepsFactory(configuration(), new EjemploCalculadora());  
    }  
  
    @Override  
    protected List<String> storyPaths() { return Arrays.asList("./Math.story"); }  
}
```

CONFIGURACIÓN DE HISTORIAS

```
public class EjemploCalculadora extends Steps {  
    int x;  
  
    @Given("una variable x con valor $value")  
    public void dado_un_valor_x(@Named("value") int value) {  
        x = value;  
    }  
  
    @When("multiplico x por $value")  
    public void cuando_multiplico_x_por(@Named("value") int value) {  
        x = Calculadora.multiplica(x,value);  
    }  
  
    @When("divido x por $value")  
    public void cuando_divido_x_por(@Named("value") int value) {  
        x = Calculadora.divide(x,value);  
    }  
  
    @Then("x deberia ser igual a $value")  
    public void entonces_x_deveria_ser(@Named("value") int value) {  
        if (value != x)  
            throw new RuntimeException("x is " + x + ", but should be " + value);  
    }  
}
```

PRUEBAS SOFTWARE: TEORÍA

4.4 Caso real: Resolución de incidencias



RESOLUCIÓN DE INCIDENCIAS EN SHIFTS



Cambios de turno asociados

Se ha producido un error inesperado al intentar saldar la solicitud.

Bad Request

Solicitante	Fecha Sol.	Receptor	Fecha A/I	Acciones
ESCANELLAS BUENAVENTURA, RAFAEL	08/05/2015 19:04	SANTANDREU BARQUERO, ANTONIO	09/05/2015 01:41 18:00 - 20:00	
ESCANELLAS BUENAVENTURA, RAFAEL	19/09/2014 15:12	SANTANDREU BARQUERO, ANTONIO	20/09/2014 01:33 18:30 - 21:30	
ESCANELLAS BUENAVENTURA, RAFAEL	11/07/2014 17:52	RAMIS PERELLO, JORGE	11/07/2014 21:40 18:00 - 20:00	

RESOLUCIÓN DE INCIDENCIAS EN SHIFTS (II)



```
@Test
public void verifco_acumulacion_deuda_simple_para_dos_cambios_de_turno(){
    realizaSolicitudesCTConDeudas(1,1);

    List<SolicitudIntercambioTurno> solicitudesDeudaPendiente = deudasService.getCTNoSaldadosFrom(DATA.solicitado);

    Assert.assertEquals(2F, solicitudesDeudaPendiente.get(0).getEmpleadoSolicitado().getBalanceCambioTurno(),getBalanceHorasAbsoluto());
    Assert.assertEquals(2F, solicitudesDeudaPendiente.get(0).getEmpleadoSolicitante().getBalanceCambioTurno(),getBalanceHorasAbsoluto());
}

@Test
public void verifco_el_saldo_de_horas_de_la_acumulacion_de_deuda_simple() {
    realizaSolicitudesCTConDeudas(1,1);

    List<SolicitudIntercambioTurno> solicitudesDeudaPendiente = deudasService.getSolicitudesConDeudasPendientes(DATA.solicitante);
    solicitudesDeudaPendiente.get(0).setDeudaSaldada(true);
    saldoDeudasService.actualizaBalanceAbsolutoEmpleadosInvolucrados(solicitudesDeudaPendiente.get(0));

    solicitudesDeudaPendiente = deudasService.getCTNoSaldadosFrom(DATA.solicitado);
    Assert.assertEquals(1F, solicitudesDeudaPendiente.get(0).getEmpleadoSolicitado().getBalanceCambioTurno(),getBalanceHorasAbsoluto());
    Assert.assertEquals(1F, solicitudesDeudaPendiente.get(0).getEmpleadoSolicitante().getBalanceCambioTurno(),getBalanceHorasAbsoluto());
}
```

RESOLUCIÓN DE INCIDENCIAS EN SHIFTS (III)



```
private void realizaSolicitudesCTConDeudas( int... deudas ) {
    setMockedCambioTurnoValidatorService();

    for(int deuda : deudas) {
        solicitudCTSinInversionDeBloquesYDeuda(deuda);
        solicitudesIntercambioTurnoService.solicitadoAceptaIntercambioTurno(DATA.solicitud, DATA.solicitado);
    }
}
```

RESOLUCIÓN DE INCIDENCIAS EN SHIFTS (III)



```
@Test
public void para_el_saldo_de_deudas_tengo_en_cuenta_las_deudas_comensadas_Para_dos_CT_con_deudas_con_deuda_1_y_menos2_tras_saldar_la_primera_la_deuda_absoluta_sera_2() {
    realizaSolicitudesCTConDeudas(1, -2);
    List<SolicitudIntercambioTurno> solicitudesDeudaPendiente = deudasService.getSolicitudesConDeudasPendientes(DATA.solicitante);

    saldoDeudaDeSolicitudConValor(solicitudesDeudaPendiente, 1);
    solicitudesDeudaPendiente = deudasService.getSolicitudesConDeudasPendientes(DATA.solicitante);

    Assert.assertEquals(2F, solicitudesDeudaPendiente.get(0).getEmpleadoSolicitado().getBalanceCambioTurno().getBalanceHorasAbsoluto());
    Assert.assertEquals(2F, solicitudesDeudaPendiente.get(0).getEmpleadoSolicitante().getBalanceCambioTurno().getBalanceHorasAbsoluto());
}

@Test
public void para_el_saldo_de_deudas_tengo_en_cuenta_las_deudas_comensadas_Para_3_CT_con_deudas_con_deuda_1_y_menos2_y_menos1_tras_saldar_la_primera_la_deuda_absoluta_sera_3() {
    realizaSolicitudesCTConDeudas(1, -2, -1);
    List<SolicitudIntercambioTurno> solicitudesDeudaPendiente = deudasService.getSolicitudesConDeudasPendientes(DATA.solicitante);

    saldoDeudaDeSolicitudConValor(solicitudesDeudaPendiente, 1);
    solicitudesDeudaPendiente = deudasService.getSolicitudesConDeudasPendientes(DATA.solicitante);

    Assert.assertEquals(3F, solicitudesDeudaPendiente.get(0).getEmpleadoSolicitado().getBalanceCambioTurno().getBalanceHorasAbsoluto());
    Assert.assertEquals(3F, solicitudesDeudaPendiente.get(0).getEmpleadoSolicitante().getBalanceCambioTurno().getBalanceHorasAbsoluto());
}
```



BLOQUES DEL CURSO

1. Introducción

- Informe CHAOS
- Código limpio



2. Test y código limpio

- Responsabilidades
- Código limpio



3. JUnit

- xUnit
- API



4. Pruebas Software: Teoría

- Historia
- Tipología
- Escritura



TDD

- Descripción
- Algoritmo TDD
- BDD, DDD, ADD, TDD



DOBLES DE PRUEBA

- Tipos de dobles
- Usos
- Mockito



TEST DE CLIENTE Y DE SISTEMA

- Cuándo y cómo probar el cliente
- Jasmine
- Selenium



IC Y HERRAMIENTAS DE CALIDAD Y VALIDACIÓN

- Definiciones
- Empezando..
- Jenkins
- JMeter

Manifiesto ágil:

- Individuos e interacciones sobre procesos y herramientas.
- Software que funciona sobre documentación exhaustiva.
- Colaboración con el cliente sobre negociación de contratos.
- Responder ante el cambio sobre un seguimiento de un plan.

Los test software:

- Generan documentación dinámicamente.
- Documentan el código (me dicen cómo y para qué funciona).
- **LOS TEST SIRVEN PARA CREAR CÓDIGO LIMPIO. CALIDAD DE SOFTWARE.** (Ejecuta las pruebas, expresa la intención del programador=> Naming , No contiene duplicados minimiza el número de clases y métodos)

JUnit:

- Assert.assertEquals()
- Expected exceptions.
- Assert.assertThat(<value>, <Matcher>).
- @Before, @After, @BeforeClass, @AfterClass.

Los test software:

- Tipos test:
 - Unitarios, integración y de sistema.
 - Aceptación y funcionales.
- El código de prueba es tan importante como el de producción.
- Características de los test: LEGIBILIDAD, LEGIBILIDAD, LEGIBILIDAD
- FIRST
- AAA

FLUJO DISEÑO BDD + TDD

Powered by
autentia



TDD

5.1 Descripción



- **Implementación de las funciones justas** que el cliente necesita y **no más**.
- **Minimización del número de defectos** que llegan al software en fase de producción.
- **Producción de software modular**, altamente **reutilizable** y preparado para el cambio.



TDD

Asegurar que aplicamos la solución software correcta a unos requisitos definidos.

- La **calidad** del software aumenta
- Conseguimos código altamente **reutilizable**
- El **trabajo en equipo** se hace más fácil.
- Nos permite **confiar** más en nuestros compañeros.
- Multiplica la **comunicación** entre los miembros del equipo.
- Las personas encargadas de la garantía de calidad adquieren un rol más inteligente e interesante.
- Escribir el ejemplo (test) antes que el código nos obliga a **escribir el mínimo de funcionalidad necesaria**, evitando sobrediseñar.
- Cuando revisamos un proyecto desarrollado mediante TDD, nos damos cuenta de que los tests son la mejor **documentación técnica** que podemos consultar a la hora de entender qué misión cumple cada pieza del puzzle.

* Definidos por Kent Beck en uno de sus [libros](#)

EJEMPLOS PARA TDD DE CLIENTE

estamos encantados de escuchar que hay nuevas necesidades, ya que a fin de cuentas es un buen indicio de un uso cotidiano de Shifts.

El alcance de los dos puntos que indicas variaría ligeramente en función de las implicaciones y la dimensión que adquiera el cambio, concretando.

1) Libres + margen

En relación al primer punto, te rogaría que nos confirmases si la siguiente solución responde a vuestras necesidades.

A la hora de solicitar un CT para un empleado dado (obviaremos a su compañero en el ejemplo) aplicaremos la modificación **exclusivamente a la regla de cambio de** que el empleado trabaja (tiene turno).

Actualmente esta regla se valida exclusivamente en función de un parámetro único, descanso entre turnos.

Con el nuevo cambio este parámetro se complementaría con un segundo parámetro "margen de varianza" o como quiera llamarse.

A partir de ahora para que esa misma regla valide, verificaremos que si el descanso efectivo es menor que el límite pero mayor o igual a (límite - margen) entonces te

Esta segunda oportunidad será validada cuando ese mismo empleado descansen el descanso límite + el diferencial anterior (puede ser menor o igual que el margen de

Si esta casuística se da para, por ejemplo el día de antes y el de después, entonces la regla no validará.

Ejemplos:

Descanso entre turnos límite: 10 horas (configurable)

Margen de varianza: 2 horas (configurable)

Juan realiza un CT de 2 días (Martes y Miércoles), el hipotético **resultado final** queda con los siguientes descansos entre turnos:

CASO 1 -> La nueva regla valida

L - M: 12 horas

M - X: 12 horas

X - J: 8 horas

J - V: 10 horas

V - S: 10 horas

CASO 2 -> La nueva regla valida

L - M: 10 horas

M - X: 10 horas

X - J: 8 horas

J - V: 12 horas

V - S: 12 horas

CASO 3 -> La nueva regla valida

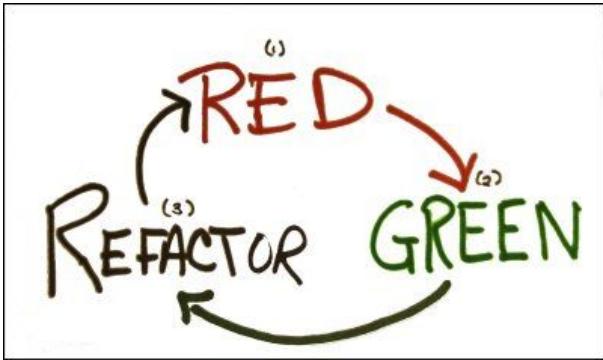
L - M: 10 horas

5

TDD

5.2 Algoritmo TDD





- TODO LIST
- No se puede tocar el código de producción mientras no haya un test. Escribir la especificación del requisito **[ROJO]**, esto nos hace pensar en la API.
- Implementar el código según el ejemplo. Al principio lo más rápido posible, imaginándonos que tuviésemos un par de minutos. **[VERDE]**
- REFACTORIZAR tranquilamente, limpiar el código. **[NEGRO]**

- LOS CAMBIOS son una constante en los desarrollos.
- Somos demasiado optimistas a la hora de resolver problemas.
- Nunca podemos tener en cuenta todas las relaciones e implicaciones => En todos los proyectos TODO EL MUNDO COMETE ERRORES.



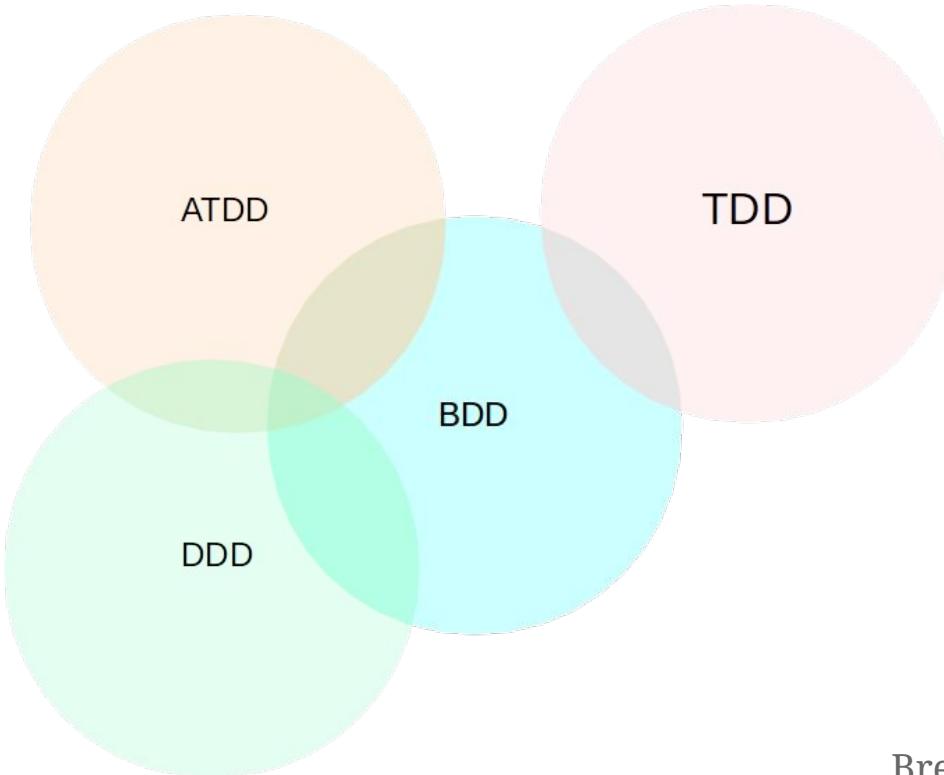
TDD

Es una guía para producir código limpio

TDD

5.3 ATDD, BDD, TDD, DDD !!!???

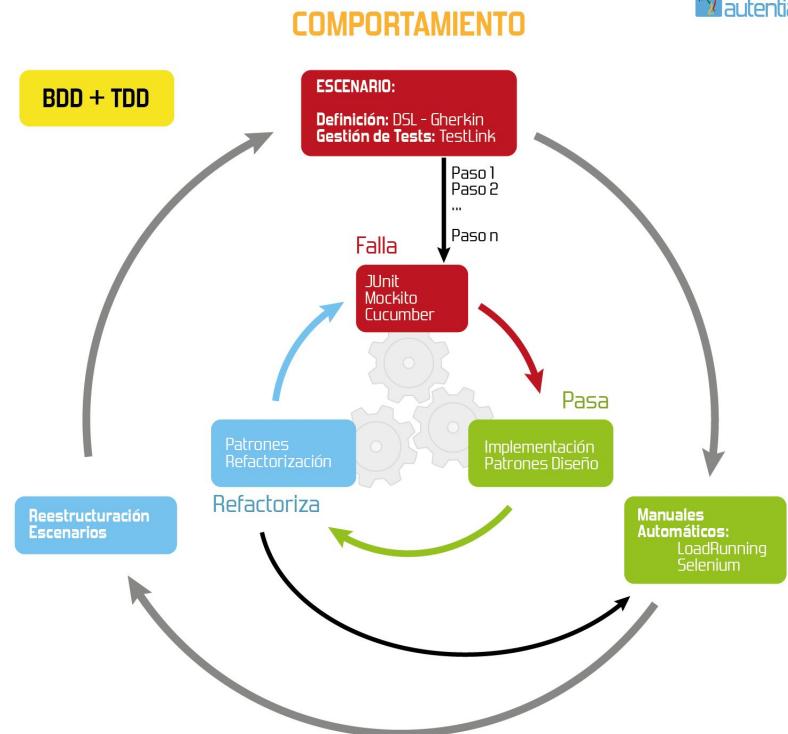




Breve digresión DDD...

FLUJO DISEÑO BDD + TDD

Powered by
autentia



No se trata de reemplazar toda la documentación, sino los requisitos. Los requisitos son historias de usuario (ejemplos)

- Dado (Given), un contexto inicial.
- Cuando (When), un evento se produce.
- Entonces (Then), aseguro algunos resultados

EJEMPLO ATDD

```
* Created by rvalero on 11/06/15.  
*/  
  
public class EjemploCalculadora extends Steps {  
    int x;  
  
    @Given("una variable x con valor $value")  
    public void dado_un_valor_x(@Named("value") int value) { x = value; }  
  
    @When("multiplico x por $value")  
    public void cuando_multiplico_x_por(@Named("value") int value) { x = Calculadora.multiplica(x,value); }  
  
    @When("divido x por $value")  
    public void cuando_divido_x_por(@Named("value") int value) { x = Calculadora.divide(x,value); }  
  
    @Then("x deberia ser igual a $value")  
    public void entonces_x_deveria_ser(@Named("value") int value) {  
        if (value != x)  
            throw new RuntimeException("x is " + x + ", but should be " + value);  
    }  
}
```

Narrative:
In order to probar una calculadora

Scenario: 2 x 2

Given una variable x con valor 2
When multiplico x por 2
Then x deberia ser igual a 4

Scenario: 3 x 3

Given una variable x con valor 3
When multiplico x por 3
Then x deberia ser igual a 9

“Como un cliente, quiero sacar dinero del cajero automático, de modo que no necesite pasar tiempo en la cola.”

- **Dado** que la cuenta posee crédito. Y la tarjeta es válida. Y el cajero automático tiene dinero.
- **Cuando** el cliente pide dinero.
- **Entonces** asegure que la cuenta sea debitada. Y asegure que el dinero sea entregado. Y asegure que la tarjeta sea devuelta.

“

El TDD puro ayuda a los desarrolladores de software a producir código de mejor calidad y facilidad de mantenimiento. El problema es que los clientes rara vez se interesan en el código. Ellos quieren que los sistemas sean más productivos y generen retornos sobre la inversión. El ATDD ayuda a coordinar los proyectos de software de forma de entregar lo que el cliente desea.

Koskela, en su libro “Test Driven: Practical TDD and Acceptance TDD for Java Developers”,

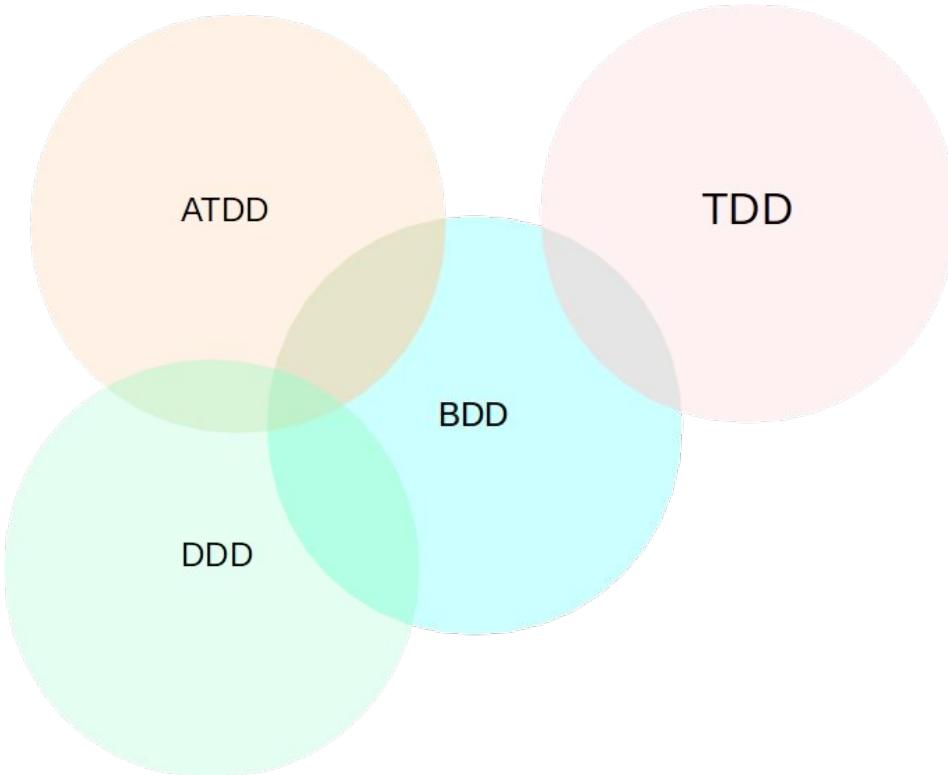


- Propiedad de los clientes.
- Escrito en conjunto con los clientes, desarrolladores y analistas de prueba.
- Sobre el Qué y no sobre el Cómo.
- Expresada en **lenguaje de dominio** del problema.
- Conciso, preciso y sin ambigüedades.

- **AUTOMATIZADA**
 - Documentada.
 - Siempre disponible.

- **PASOS:**
 - Historia
 - Pruebas de aceptación
 - Automatización
 - Funcionalidad

BDD



TDD

5.3 IS TDD DEAD?





<http://martinfowler.com/articles/is-tdd-dead/>

**Kent Beck**

@KentBeck



Following

tdd is not "alive" or "dead". it is subject to tradeoffs, including risk of api changes, skill of practitioner, and existing design.

TDD

5.4 StringWrap





BLOQUES DEL CURSO

1. Introducción

- Informe CHAOS
- Código limpio



TDD

- Descripción
- Algoritmo TDD
- BDD, DDD, ADD, TDD



2. Test y código limpio

- Responsabilidades
- Código limpio



DOBLES DE PRUEBA

- Tipos de dobles
- Usos
- Mockito



3. JUnit

- xUnit
- API

4. Pruebas Software: Teoría

- Historia
- Tipología
- Escritura



IC Y HERRAMIENTAS DE CALIDAD Y VALIDACIÓN

- Definiciones
- Empezando..
- Jenkins
- JMeter



TEST DE CLIENTE Y DE SISTEMA

- Cuándo y cómo probar el cliente
- Jasmine
- Selenium

Manifiesto ágil:

- Individuos e interacciones sobre procesos y herramientas.
- Software que funciona sobre documentación exhaustiva.
- Colaboración con el cliente sobre negociación de contratos.
- Responder ante el cambio sobre un seguimiento de un plan.

Los test software:

- Generan documentación dinámicamente.
- Documentan el código (me dicen cómo y para qué funciona).
- **LOS TEST SIRVEN PARA CREAR CÓDIGO LIMPIO. CALIDAD DE SOFTWARE.** (Ejecuta las pruebas, expresa la intención del programador=> Naming , No contiene duplicados minimiza el número de clases y métodos)

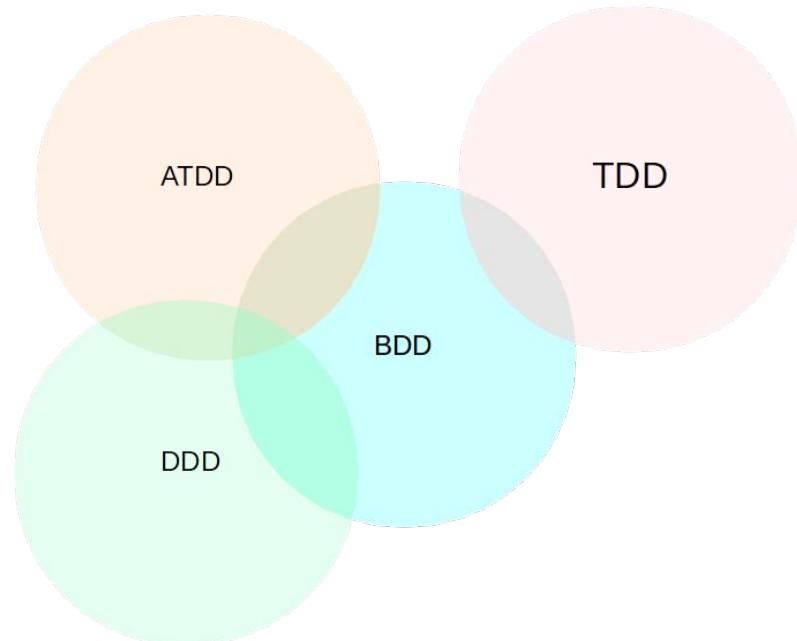
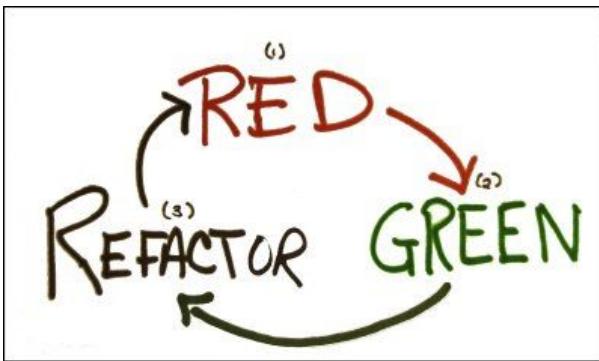
JUnit:

- Assert.assertXXX()
- Expected exceptions.
- Assert.assertThat(<value>, <Matcher>).
- @Before, @After, @BeforeClass, @AfterClass.

Los test software:

- Tipos test:
 - Unitarios, integración y de sistema.
 - Aceptación y funcionales.
- El código de prueba es tan importante como el de producción.
- Características de los test: LEGIBILIDAD, LEGIBILIDAD, LEGIBILIDAD
- FIRST
- AAA

PUNTOS CLAVE DEL BLOQUE 5



6

DOBLES DE PRUEBA

6.1 Introducción



DEFINICIÓN



EASYMOCK



- El código del test puede llegar a ser difícil de leer.
- El test corre el riesgo de volverse frágil si conoce demasiado bien el interior del SUT.

DOBLES DE PRUEBA

6.2 Tipos de dobles



- Dummy
- Fake
- Stub
- Mock
- Spy

“

*Ni Kent Beck ni Martin Fowler utilizan
frameworks de Mocks*



DOBLES DE PRUEBA

6.3 Mockito



CREACIÓN DE UN MOCK

```
Flower flowerMock = Mockito.mock(Flower.class);
```

```
@Mock  
private Flower flowerMock;
```

STUB DE UN VALOR DE RETORNO DE UN MÉTODO

```
public class Mockito {  
  
    private static final java.lang.Integer TEST_NUMBER_OF_LEAFS = ...;  
  
    @Test  
    public void test() {  
        Flower flowerMock = mock(Flower.class);  
        when(flowerMock.getNumberOfLeafs()).thenReturn(TEST_NUMBER_OF_LEAFS);  
        int numberOfLeafs = flowerMock.getNumberOfLeafs();  
        assertThat(numberOfLeafs, is(TEST_NUMBER_OF_LEAFS));  
    }  
  
    private class Flower {  
        private int numberOfLeafs;  
  
        public int getNumberOfLeafs() {  
            return numberOfLeafs;  
        }  
    }  
}
```

DIFERENTES COMPORTAMIENTOS

Método	Descripción
<code>thenReturn(T valueToBeReturned)</code>	Retorna el valor dado
<code>thenThrow(Throwable toBeThrown)</code> <code>thenThrow(Class toBeThrown)</code>	Lanza una excepción dada
<code>then(Answer answer)</code> <code>thenAnswer(Answer answer)</code>	Usa una respuesta creada
<code>thenCallRealMethod()</code>	Llama al método del objeto original (spy)

```
public class Mockito {

    private static final int TEST_NUMBER_OF_LEAFS = 1;

    @Test
    public void test() {
        //given
        Flower flowerMock = mock(Flower.class);
        given(flowerMock.getNumberOfLeafs()).willReturn(TEST_NUMBER_OF_LEAFS);
        //when
        int numberofLeafs = flowerMock.getNumberOfLeafs();
        //then
        assertEquals(numberofLeafs, TEST_NUMBER_OF_LEAFS);
    }

    private class Flower {
        private int numberofLeafs;

        public int getNumberOfLeafs() {
            return numberofLeafs;
        }
    }
}
```

VERIFICACIÓN DE ARGUMENTOS

```
private static final int VALUE_FOR_WANTED_ARGUMENT = 1;
private static final int WANTED_DATE = 1;
private static final int ANY_OTHER_DATE = 1;

@Test
public void test() {
    Item schedulerMock= mock(Item.class);
    given(schedulerMock.metodo(WANTED_DATE)).willReturn(VALUE_FOR_WANTED_ARGUMENT);

    int numberForWantedArgument = schedulerMock.metodo(WANTED_DATE);
    int numberForAnyOtherArgument = schedulerMock.metodo(ANY_OTHER_DATE);

    assertEquals(numberForWantedArgument, VALUE_FOR_WANTED_ARGUMENT);
}

private class Item {

    public int metodo(int n) {
        return n;
    }
}
```

FLEXIBILIDAD EN LOS MÉTODOS DE ENTRADA

```
public class Mockito {

    private static final int VALUE_FOR_WANTED_ARGUMENT = 1;
    private static final int WANTED_DATE = 1;
    private static final int ANY_OTHER_DATE = 1;

    @Test
    public void test() {
        Item schedulerMock= mock(Item.class);
        given(schedulerMock.metodo(anyInt())).willReturn(VALUE_FOR_WANTED_ARGUMENT);

        int numberForWantedArgument = schedulerMock.metodo(WANTED_DATE);
        int numberForAnyOtherArgument = schedulerMock.metodo(ANY_OTHER_DATE);

        assertEquals(numberForWantedArgument, VALUE_FOR_WANTED_ARGUMENT);
    }

    private class Item {
        public int metodo(int n) {
            return n;
        }
    }
}
```

DIFERENTES COMPORTAMIENTOS

Nombre	Reglas de coincidencia
any(), any(Class clazz)	Cualquier objeto o null
anyBoolean(), anyByte(), anyChar(), anyDouble(), anyFloat(), anyInt(), anyLong(), anyShort(), anyString()	Cualquier objeto del tipo dado o null
anyCollection(), anyList(), anyMap(), anySet()	Respectivamente cualquier tipo de colección
anyCollectionOf(Class clazz), anyListOf(Class clazz), anyMapOf(Class clazz), anySetOf(Class clazz)	Respectivamente cualquier colección del tipo dado
eq(T value)	Cualquier objeto igual al dado (equals)
isNull, isNull(Class clazz)	null

DIFERENTES COMPORTAMIENTOS II

Nombre	Reglas de coincidencia
isA(Class clazz)	Cualquier objeto que implemente la clase dada
refEq(T value, String... excludeFields)	Cualquier objeto que sea igual al dado usando reflexión
matches(String regex)	String que coincide con la expresión regular pasada como parámetro
startsWith(string), endsWith(string), contains(string) for a String class	String que cumpla las reglas
aryEq(PrimitiveType value[]), aryEq(T[] value)	Cualquier array que sea igual al dado (tiene la misma longitud y cada elemento es igual)
cmpEq(Comparable value)	Cualquier objeto que sea igual al dado usando el método compareTo()

DIFERENTES COMPORTAMIENTOS III

Nombre	Reglas de coincidencia
gt(value), geq(value), lt(value), leq(value)	Cualquier valor mayor, mayor o igual, igual, menor o menor o igual para tipos primitivos u objetos que implementan Comparable
argThat(org.hamcrest.Matcher matcher)	Los objetos que satisfagan el Matcher pasado por parámetro
booleanThat(Matcher matcher), byteThat(matcher), charThat(matcher), doubleThat(matcher), floatThat(matcher), intThat(matcher), longThat(matcher), shortThat(matcher)	Cualquier objeto del tipo dado que satisfaga el matcher
and(first, second), or(first, second), not(first)	Para combinar diferentes matchers

VERIFICACIÓN DE COMPORTAMIENTO

```
@Test  
public void test() {  
    WaterSource waterSourceMock = mock(WaterSource.class);  
    waterSourceMock.doSelfCheck();  
    verify(waterSourceMock).doSelfCheck();  
}  
  
private class WaterSource {  
    public void doSelfCheck() {  
    }  
}
```

DIFERENTES COMPORTAMIENTOS III

Nombre	Verifica que el método fué
<code>times(int wantedNumberOfInvocations)</code>	llamado un numero exacto de veces
<code>never()</code>	nunca llamado
<code>atLeastOnce()</code>	llamado al menos una vez
<code>atLeast(int minNumberOfInvocations)</code>	llamado al menos un número exacto de veces
<code>atMost(int maxNumberOfInvocations)</code>	llamado un número máximo de veces
<code>only()</code>	la única llamada de un método de un mock
<code>timeout(int millis)</code>	ejecutado en un intervalo de tiempo especificado

VERIFICACIÓN DE ARGUMENTOS

Nombre	Verifica que el método fué
<code>times(int wantedNumberOfInvocations)</code>	llamado un numero exacto de veces
<code>never()</code>	nunca llamado
<code>atLeastOnce()</code>	llamado al menos una vez
<code>atLeast(int minNumberOfInvocations)</code>	llamado al menos un número exacto de veces
<code>atMost(int maxNumberOfInvocations)</code>	llamado un número máximo de veces
<code>only()</code>	la única llamada de un método de un mock
<code>timeout(int millis)</code>	ejecutado en un intervalo de tiempo especificado

- @Mock
- @Spy
- @Captor
- @InjectMocks

- Mockear clases finales
- Mockear enums
- Mockear métodos final
- Mockear métodos estáticos
- Mockear métodos privados
- Mockear los métodos
hashCode() o equals()

TEST DE CLIENTE Y SISTEMA

6.1 CLIENTE



TEST DE CLIENTE



WKARMA



simple, flexible, fun

“

¿Por qué probar el código de cliente?



TEST DE CLIENTE Y SISTEMA

6.2 CLIENTE: Jasmine



```
6
7     describe("Una suite", function() {
8         it("contiene especificaciones", function() {
9             expect(true).toBe(true);
10        });
11
12        it("que deben contener expectativas", function() {
13            expect(false).toBe(false);
14        });
15    });
16
17
```

beforeEach, afterEach, beforeAll y afterAll

```
4  describe("A spec using beforeAll and afterAll", function() {
5    var foo;
6
7    beforeEach(function() {
8      foo = 1;
9    });
10
11   afterEach(function() {
12     foo = 0;
13   });
14
15   it("sets the initial value of foo before specs run", function() {
16     expect(foo).toEqual(1);
17     foo += 1;
18   });
19
20   it("does not reset foo between specs", function() {
21     expect(foo).toEqual(2);
22   });
23 });
24 });
25
26
27
```

beforeEach, afterEach, beforeAll y afterAll (||)

```
1
2
3
4
5  describe("A spec", function() {
6    beforeEach(function() {
7      this.foo = 0;
8    });
9
10   it("can use the `this` to share state", function() {
11     expect(this.foo).toEqual(0);
12     this.bar = "test pollution?";
13   });
14
15   it("prevents test pollution by having an empty `this` created for the next spec", function() {
16     expect(this.foo).toEqual(0);
17     expect(this.bar).toBe(undefined);
18   });
19 });
20
21
```

Función	Descripción
<code>toBe(null false true)</code>	Comparación mediante '==='
<code>toEqual(value)</code>	Comparación para literales, variables y objetos
<code>toMatch(regex string)</code>	Comparación para expresiones regulares
<code>toBeUndefined() toBeDefined()</code>	Comparación contra 'undefined'
<code>toBeNull()</code>	Comparación contra null
<code>toBeTruthy() toBeFalsy</code>	Comparación booleana con casting implícito
<code>toContain(string)</code>	Para encontrar elementos de un array

- Dobles de pruebas
- Librerías para llamadas asíncronas
(ajax)

TEST DE CLIENTE Y SISTEMA

6.3 SISTEMA: Selenium





“Práctica 8: Selenium”

Test de sistema con Selenium IDE y configuración para ejecución desde JUnit





1. INTRODUCCIÓN

- Informe CHAOS
- Código limpio



5. TDD

- Descripción
- Algoritmo TDD
- BDD, DDD, ADD, TDD



2. TEST Y CÓDIGO LIMPIO

- Responsabilidades
- Código limpio



6. DOBLES DE PRUEBA

- Tipos de dobles
- Usos
- Mockito

BLOQUES DEL CURSO



3. JUnit

- xUnit
- API



4. PRUEBAS SOFTWARE: TEORÍA

- Historia
- Tipología
- Escritura



7. TEST DE CLIENTE Y DE SISTEMA

- Cuándo y cómo probar el cliente
- Jasmine
- Selenium



8. IC Y HERRAMIENTAS DE CALIDAD Y VALIDACIÓN

- Definiciones
- Empezando..
- Jenkins
- JMeter

Manifiesto ágil:

- Individuos e interacciones sobre procesos y herramientas.
- Software que funciona sobre documentación exhaustiva.
- Colaboración con el cliente sobre negociación de contratos.
- Responder ante el cambio sobre un seguimiento de un plan.

Los test software:

- Generan documentación dinámicamente.
- Documentan el código (me dicen cómo y para qué funciona).
- **LOS TEST SIRVEN PARA CREAR CÓDIGO LIMPIO. CALIDAD DE SOFTWARE.** (Ejecuta las pruebas, expresa la intención del programador=> Naming , No contiene duplicados minimiza el número de clases y métodos)

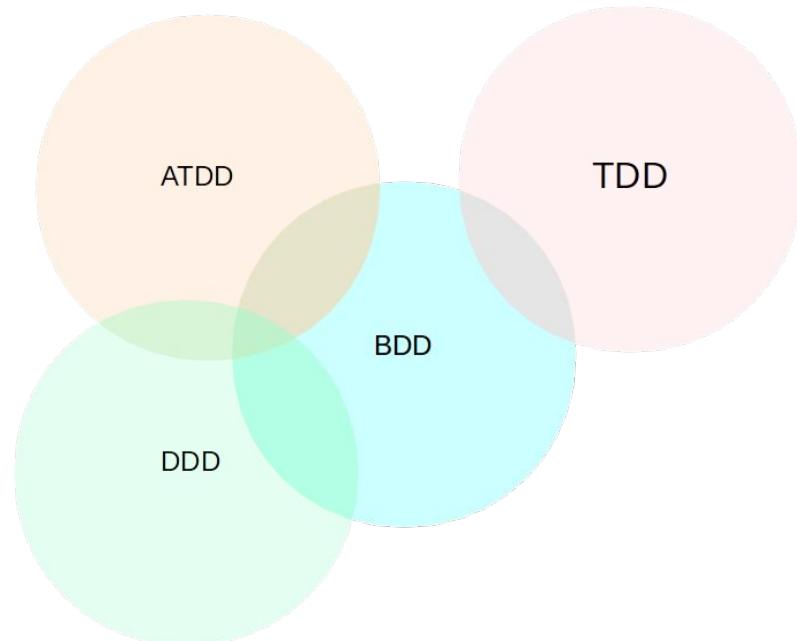
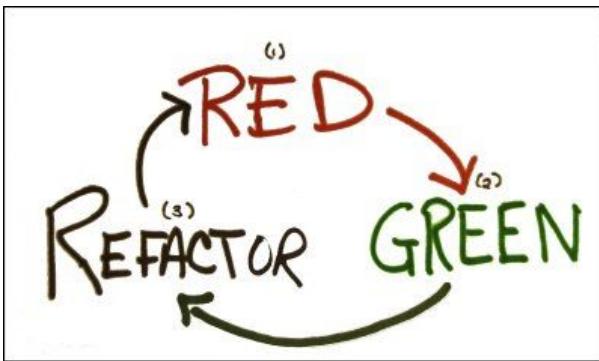
JUnit:

- Assert.assertXXX()
- Expected exceptions.
- Assert.assertThat(<value>, <Matcher>).
- @Before, @After, @BeforeClass, @AfterClass.

Los test software:

- Tipos test:
 - Unitarios, integración y de sistema.
 - Aceptación y funcionales.
- El código de prueba es tan importante como el de producción.
- Características de los test: LEGIBILIDAD, LEGIBILIDAD, LEGIBILIDAD
- FIRST
- AAA

PUNTOS CLAVE DEL BLOQUE 5



Dobles de prueba

- Probar cada pieza software (SUT) de manera aislada
- Mockito

Test de sistema

- Selenium

Test de cliente

- Probar el cliente de manera aislada (no su integración)
- Jasmine

INTEGRACIÓN CONTINUA, CALIDAD DEL SOFTWARE, PRUEBAS DE ACEPTACIÓN CON JMETER

6.1 Pruebas de aceptación



- Carencia de un proceso de desarrollo que integre las actividades de las pruebas
- Sobrevaloración de la automatización de las pruebas como proceso inmediato
- No “rentabilizar” el esfuerzo invertido en pruebas

- Una PA tiene como objetivo demostrar al cliente el cumplimiento de un **requisito** software
- Una PA:
 - Describe un **escenario** (secuencia de pasos) de un sistema desde la perspectiva del cliente
 - Puede estar asociada a requisitos funcionales o no funcionales
 - Un requisito tiene una o más PAs
 - Las PAs cubren TODOS los escenarios
- Given/Then/When es un enfoque

- Adicional a su propósito fundamental, las PAs pueden rentabilizarse usándose para:
 - Obligar a definir requisitos que sean verificables
 - Valorar adecuadamente el esfuerzo asociado a la incorporación de un requisito
 - Negociar con el cliente el alcance del sistema
 - Planificar el desarrollo iterativo e incremental del sistema
 - Guiar a los desarrolladores ¾ Identificar oportunidades de reutilización

“El proceso de desarrollo debe estar dirigido por los requisitos”. Obvio puesto que los requisitos son el objetivo a cumplir, sin embargo, ...

- ¿Popularmente cómo se especifican los requisitos?
 - Textualmente
 - UML (Diagramas de Casos de Uso y otros diagramas)
 - Plantillas o fichas
 - Interfaces de usuario (bocetos)
 - Combinación de los anteriores
- Se trata de identificar y diseñar las PAs desde los requisitos

INTEGRACIÓN CONTINUA, CALIDAD DEL SOFTWARE, PRUEBAS DE ACEPTACIÓN CON JMETER

6.2 Pruebas de aceptación con jMeter





<http://gatling.io/#/>

INTEGRACIÓN CONTINUA, CALIDAD DEL SOFTWARE, PRUEBAS DE ACEPTACIÓN CON JMETER

6.3 ¿Qué es la IC?



“

“Una práctica del desarrollo de software donde los miembros del equipo integran su trabajo con frecuencia: normalmente, cada persona integra de forma diaria, conduciendo a múltiples integraciones por día. Cada integración es comprobada por una construcción automática (incluyendo las pruebas) para detectar errores de integración tan rápido como sea posible. Muchos equipos encuentran que este enfoque conduce a la reducción significativa de problemas de integración y permite a un equipo desarrollar software cohesivo más rápidamente”

(Martin Fowler)



BUILD convertir
código fuente en
software que
funciona



¡TEST PIEZA VITAL!

JUnit



- Estructura el proyecto por cada tipo de test para lanzarlas por separado.
- Escribe también pruebas para los defectos/bugs que irán saliendo antes de corregirlos y como garantía del arreglo del fallo.

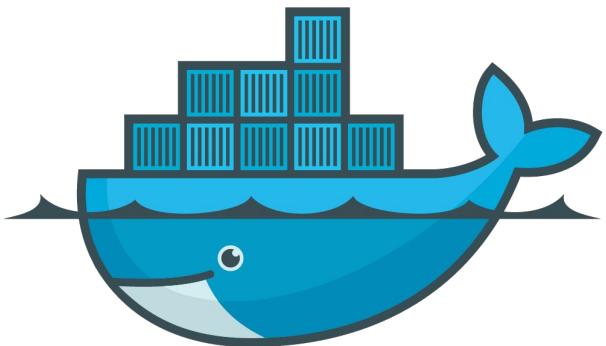
CONSTRUCCIÓN



- Despliegues demasiado complicados.
- Descubrir defectos demasiado tarde.
- Baja calidad del software

¿Qué es Docker?

Esta herramienta nos permite crear lo que ellos denominan contenedores, lo cual son aplicaciones empaquetadas auto-suficientes, muy livianas que son capaces de funcionar en prácticamente cualquier ambiente, ya que tiene su propio sistema de archivos, librerías, terminal, etc.



docker



<http://nemo.sonarqube.org/>