Práctica 5. Gestión de la calidad 1	Entregable: Captura de pantalla o ficheros con resultados
	Fecha Entrega: 12/3/20
	Equipo: Individual
	Herramientas: Maven, w3af, Tomcat

Introducción

Esta práctica es la primera dedicada al Área de Conocimiento de la Calidad. Se centra en automatizar el análisis de código, un test unitario y realizar un escaneo http.

Desarrollo de la práctica

1.-Instalación del servidor Apache Tomcat en el contenedor/equipo propio.

Ya esta instalado en la máquina virtual.

2.- Verificación

2.1.-Integrad en el proyecto de la práctica 3 de Maven *(simple*) una herramienta de análisis de código, como puede ser FindBugs, PMD o SonarQube.

Añadimos este plugin en el apartado de los plugins del pom.xml de la carpeta simple.

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>findbugs-maven-plugin</artifactId>
  <version>2.0.1</version>
  <configuration>
    <xmlOutput>true</xmlOutput>
    <!– Optional directory to put FindBugs xdoc xml report –>
    <xmlOutputDirectory>target/site</xmlOutputDirectory>
    <threshold>High</threshold>
  </configuration>
</plugin>
```

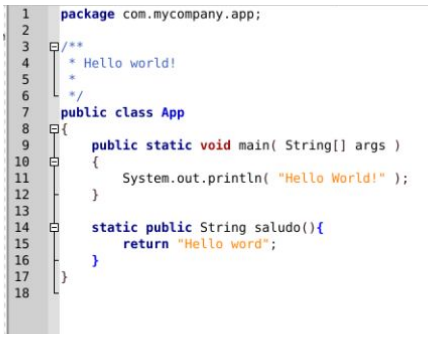
2.2.-A nivel de las especificaciones (no es necesario realizar pruebas con ambos), qué diferencias detectas entre la herramienta de análisis de código y esta otra herramienta desarrollada por el SEI: https://www.sei.cmu.edu/news-events/news/article.cfm?assetid=524804

La diferencia principal es que la herramienta del artículo está pensado para ayudar en la clasificación por importancia de las vulnerabilidades/errores encontradas.

Normalmente en proyectos grandes estas herramientas saltan con demasiados avisos y es tarea ardua identificar y examinar la importancia de cada una de ellas. Para ello esta herramienta del artículo quiere ayudar en la clasificación de las alertas.

2.3 Test Unitario: Cread un test unitario básico dentro del proyecto de la práctica 3 *(simple)* de Maven que compruebe un mensaje por consola fijo (un String).

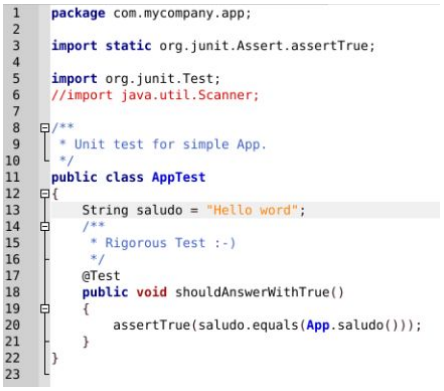
Primero cambiamos el código para la comprobación mediante strings:



```java
package com.mycompany.app;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Hello World!" );
    }

    static public String saludo(){
        return "Hello word";
    }
}
```



```java
package com.mycompany.app;

import static org.junit.Assert.assertTrue;

import org.junit.Test;
//import java.util.Scanner;

/**
 * Unit test for simple App.
 */
public class AppTest
{
    String saludo = "Hello word";
    /**
     * Rigorous Test :-)
     */
    @Test
    public void shouldAnswerWithTrue()
    {
        assertTrue(saludo.equals(App.saludo()));
    }
}
```

Ejecutamos en la terminal:

- mvn surefire:test

Si cambiamos los Strings y hacemos que no sean iguales al ejecutar el test nos da error.

2.4 Medir cobertura del proyecto tras añadir el Test Unitario anterior. Se recomienda emplear la herramienta *maven-cobertura-plugin*

Ejecutamos en terminal: *mvn cobertura:cobertura* con esto maven nos genera un reporte del código en ${project}/target/site/cobertura/index.html.

También podemos añadir al pom.xml el siguiente reporte y plugin:

```
<reporting>
  <plugins>
   <!-- Normally, we take off the dependency report, saves time. -->
   <plugin>
         <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-project-info-reports-plugin</artifactId>
         <version>2.7</version>
         <configuration>

<dependencyLocationsEnabled>false</dependencyLocationsEnabled>
         </configuration>
   </plugin>

   // integrate maven-cobertura-plugin to project site
   <plugin>
         <groupId>org.codehaus.mojo</groupId>
         <artifactId>cobertura-maven-plugin</artifactId>
         <version>2.6</version>
         <configuration>
               <formats>
                     <format>html</format>
                     <format>xml</format>
               </formats>
```

```
        </configuration>
    </plugin>

    </plugins>
</reporting
```

De este modo al ejecutar mvn site nos creará el reporte también en ${project}/site/index.html

3.-Validación

3.1-Añadid al servlet/jsp generado en la practica 3 de Maven (*simplewebapp*) una respuesta fija ante cualquier llamada http (ya sea por GET o POST). Generad e instalad el war en Tomcat y realizad un escaneo con w3af por línea de comando sobre la URL del servlet.

Aseguraros de usar la URL correcta. Probad antes con un navegador. No uséis otra URL para realizar pruebas porque podéis afectar a otros servicios y puede haber responsabilidades derivadas.

Más información en http://docs.w3af.org/en/latest/scripts.html
Como script de prueba usad el siguiente modificando la URL:
https://github.com/andresriancho/w3af/blob/master/scripts/xss_simple.w3af

NO HAY QUE HACER POR POR PROBLEMAS DE VERSIONES!!!

4.-Cumplimiento normativo

Investiga qué estándares y regulaciones pueden afectar al proyecto del *Sistema de control de contaminación* y detalla en qué cuestiones podrían afectar.

Podríamos considerar el proyecto como parte de la industria 4.0. Por lo tanto un aspecto importante serían los estándares en ciberseguridad: ISO/TC 292/WG 2 Continuidad y resiliencia, ISO/IEC JTC 1/SC 27 Técnicas de seguridad para tecnologías de la información e IEC/SC 65C/WG 13 Redes industriales.

En nuestro caso estaríamos hablando de red industrial por lo tanto deberíamos seguir los estándares arriba mencionados.

Otro estándar a tener en cuenta es la serie ISO 23570 Sistemas e integración de automatización industrial. Este estándar se basa en que con la industria 4.0 habrá muchas máquinas con muchos sensores y accionadores distintos y se trata de poner en común una base para que sean lo más universales posibles para ahorrar en costes de mantenimiento.