

¿Qué ofrece Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
 Ese apoyo que siempre quiso tener...

1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces,
 HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
 Gestor de contenidos (Alfresco)
 Aplicaciones híbridas

Tareas programadas (Quartz)
 Gestor documental (Alfresco)
 Inversión de control (Spring)

Control de autenticación y
 acceso (Spring Security)
 UDDI
 Web Services
 Rest Services
 Social SSO
 SSO (Cas)

JPA-Hibernate, MyBatis
 Motor de búsqueda empresarial (Solr)
 ETL (Talend)

Dirección de Proyectos Informáticos.
 Metodologías ágiles
 Patrones de diseño
 TDD

BPM (jBPM o Bonita)
 Generación de informes (JasperReport)
 ESB (Open ESB)



[Ver cursos que ofrece Autentia](#)



Estamos escribiendo un libro sobre la profesión informática y estas viñetas formarán parte de él. Puedes opinar en la sección [comic](#).

[Descargar comics en PDF y alta resolución](#)

Catálogo de servicios Autentia (PDF 6,2MB)



[En formato comic...](#)

Tutorial desarrollado por



Raúl Expósito Díaz

Consultor tecnológico de desarrollo de proyectos informáticos.

Puedes encontrarme en [Autentia](#)

Somos expertos en Java/J2EE

Catálogo de servicios de Autentia

[Descargar \(6,2 MB\)](#)

[Descargar en versión comic \(17 MB\)](#)

[AdictosAlTrabajo.com](#) es el Web de difusión de conocimiento de [Autentia](#).



[Catálogo de cursos](#)

Descargar este documento en formato PDF: [crap4j.pdf](#)

Fecha de creación del tutorial: 2008-01-20

Crap4j, ¿es tu código difícilmente mantenible?

1. Introducción

[Crap4j](#) es una implementación de la métrica CRAP (Change Risk Analysis and Predictions), algo que en castellano podríamos traducir como "*Análisis y Predicciones sobre los Riesgos de realizar Cambios*". Sin embargo esta es la manera "fina" de llamarlo, ya que la palabra 'crap' tiene, en inglés, un significado mucho más directo. Para hacernos una idea del significado real del término 'crap' solo tenemos que fijarnos en que el logotipo de [Crap4j](#) es un rollo de papel higiénico y que aparecen moscas alrededor.



Pero, ¿para qué sirve este plugin exactamente?. Su función es analizar código fuente y buscar aquellas clases y métodos que alguien hizo en el pasado y que, a estas alturas del desarrollo, son muy difíciles de modificar, bien sea porque el código está programado de un modo que sea difícil de leer y de entender, o bien porque es prácticamente imposible determinar cual puede ser el alcance de nuestros cambios. Si funciona bien, se suele decir que "*si funciona no lo toques*" (más que nada porque a ver quien es el valiente que se atreve a tocarlo), pero si funciona mal o queremos cambiar su modo de funcionamiento podemos tener un problema de los gordos.

Digamos que trata de localizar ese código con el que todos nos hemos encontrado alguna vez al auditar, al mantener o al continuar con un desarrollo y hemos tenido algún pensamiento *bastante* negativo sobre él. Podemos poner muchos adjetivos sobre ese código, y ninguno bueno. Por educación me limitaré a decir que es un código *difícil de mantener*.

Pero, ¿en qué se basa la métrica CRAP para evaluar un método?. Pues en la combinación de dos factores:

- **Su complejidad ciclomática**, o número de bifurcaciones únicas presentes en un método.
- **Su cobertura por pruebas unitarias**, o % del código que es ejecutado cuando se lanzan las pruebas unitarias sobre ese método.

Para obtener un índice utiliza la siguiente fórmula:

$$CRAP(m) = comp(m)^2 * (1 - cov(m)/100)^3 + comp(m)$$

Donde

- CRAP(m) es un índice que, a mayor valor, implicará mayor dificultad de mantenimiento en el método *m*
- comp(m) es el valor de la complejidad ciclomática del método *m*
- cov(m) es el % de cobertura de código del método *m* por pruebas unitarias

Valores pequeños de CRAP(m) indican que los métodos son fácilmente mantenibles, mientras que a medida que el valor aumenta, también aumenta la dificultad de mantenimiento. Para disminuir el valor de CRAP(m) debes hacer dos cosas: escribir pruebas unitarias que aseguren que cualquier cambio en el código no va a afectar al resto de la aplicación y reducir su complejidad ciclomática partiendo ese método en otros más pequeños (que también incluyan pruebas unitarias). Un método se considera difícilmente mantenible cuando su índice CRAP es mayor que 30.

Una vez vista la teoría y la razón de ser de la métrica vamos a seguir en el tutorial con la instalación del plugin de [Crap4j](#) para Eclipse, el cual os dirá si un método es difícilmente mantenible o no. [Crap4j](#) genera informes en html donde podréis ver qué métodos son los más problemáticos y cuales los menos para que podáis obrar en consecuencia y, además, puede ser lanzado desde [ant](#), con lo cual podréis lanzarlo programadamente. Si alguien se anima a escribir un tutorial sobre ello que no dude en hacerlo :-)

2. Entorno

- [Debian GNU/Linux 4.1 \(Lenny\)](#)
- [JDK 6 Update 1](#)



Web
www.adictosaltrabajo.com

Últimos tutoriales

2008-01-20
[Crap4j, ¿es tu código difícilmente mantenible?](#)

2008-01-19
[SpringIDE, plugin de Spring para Eclipse](#)

2008-01-18
[Búsqueda de dependencias para maven](#)

2008-01-18
[Icefaces, JBoss, Maven2 y EJB3: Parte 3](#)

2008-01-17
[Icefaces, JBoss, Maven2 y EJB3: Parte 2](#)

2008-01-17
[Icefaces, JBoss, Maven2 y EJB3: Parte 1](#)

2008-01-17
[Como integrar tareas Ant en Maven](#)

2008-01-16
[Ejemplo de web con ICEfaces](#)

2008-01-13
[Monitorización y profiling de aplicaciones java con VisualVM](#)

2008-01-11
[RMI y como registrar objetos en un Registry remoto](#)

Últimas ofertas de empleo

2008-01-10
[T. Información - Analista / Programador - MADRID.](#)

2008-01-08
[Otras - Ingeniería \(minas, puentes y puertos\) - SEVILLA.](#)

2007-12-28
[Comercial - Tecnología - MADRID.](#)

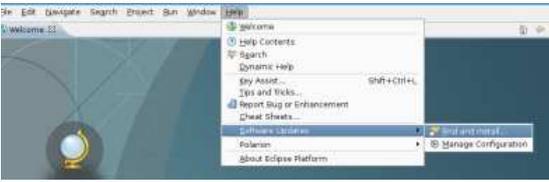
- Crap4J 1.1.6
- Eclipse 3.3 (Europa)

2007-12-28
[Comercial - Tecnología - BARCELONA.](#)

2007-12-24
[Otras Sin catalogar - SEVILLA.](#)

3. Instalación

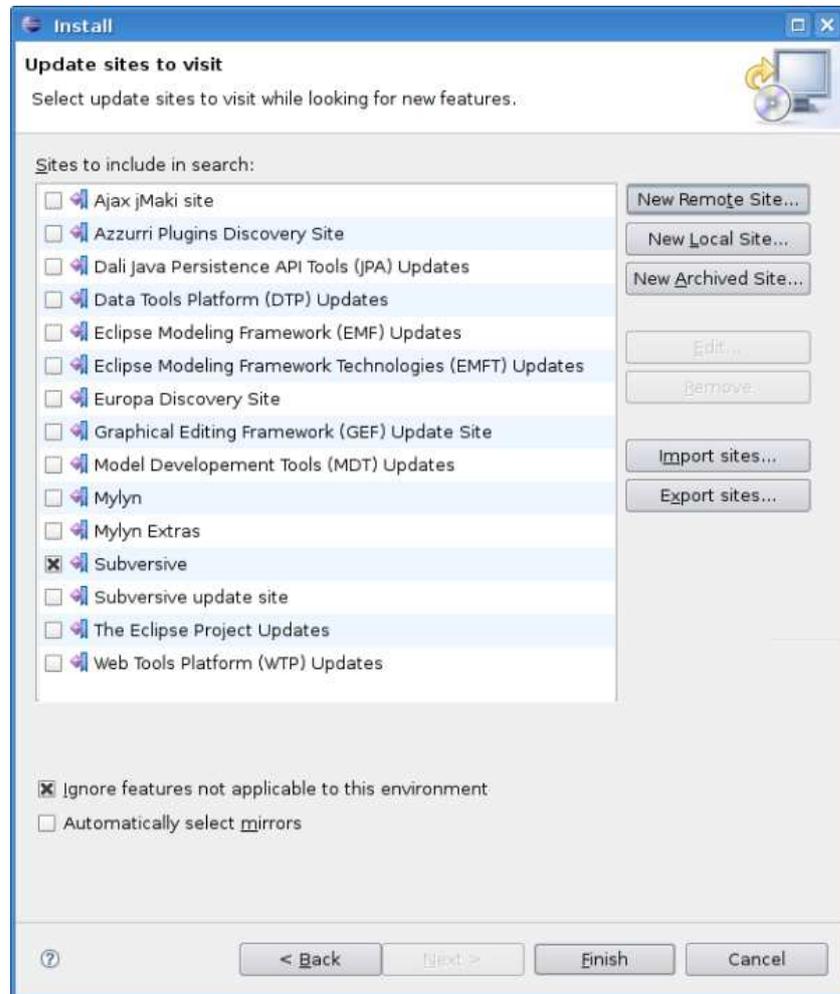
Para instalar el plugin es imprescindible que tengamos una versión de Eclipse superior a la 3.2.1. Si lo tenemos lo ejecutamos, y una vez haya terminado de cargar deberemos ir al apartado 'Help > Software Updates > Find and Install' para acceder al gestor de actualizaciones de Eclipse.



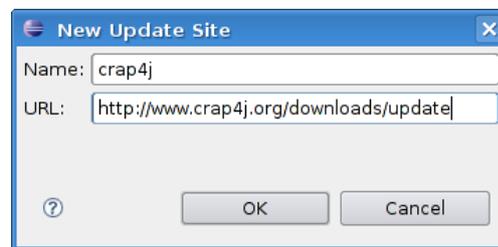
Indicamos que queremos instalar nuevas características.



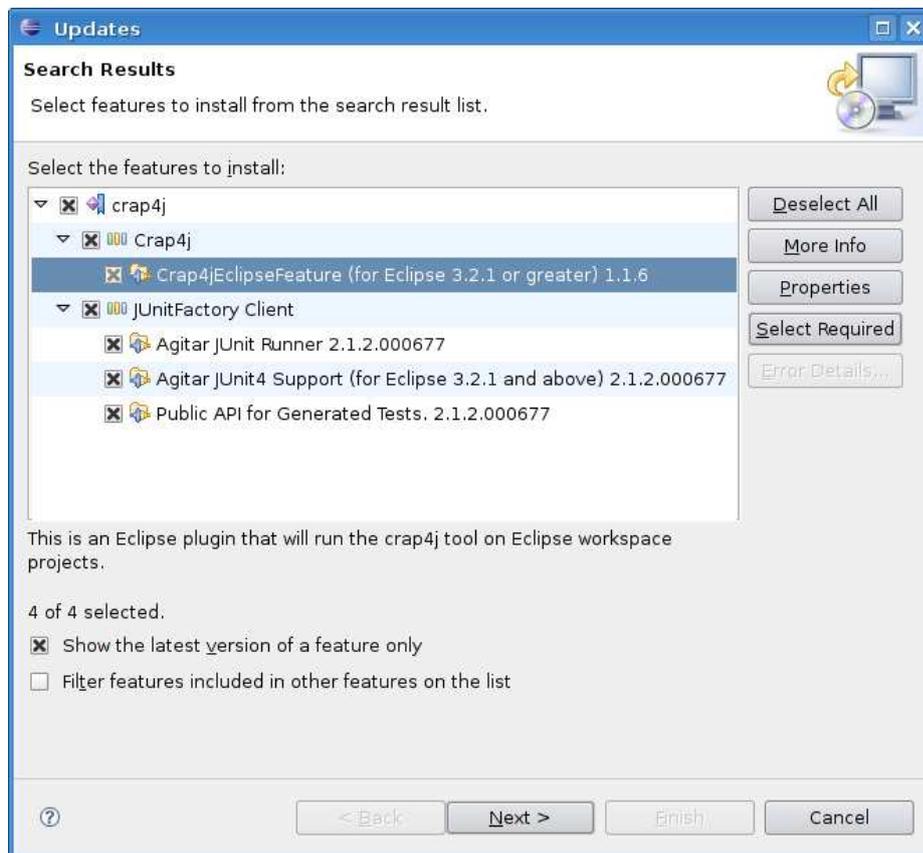
Nos aparecerá una pantalla similar a esta. Vamos a instalar el plugin de Crap4j desde un repositorio remoto, así que pinchamos sobre 'New Remote Site'



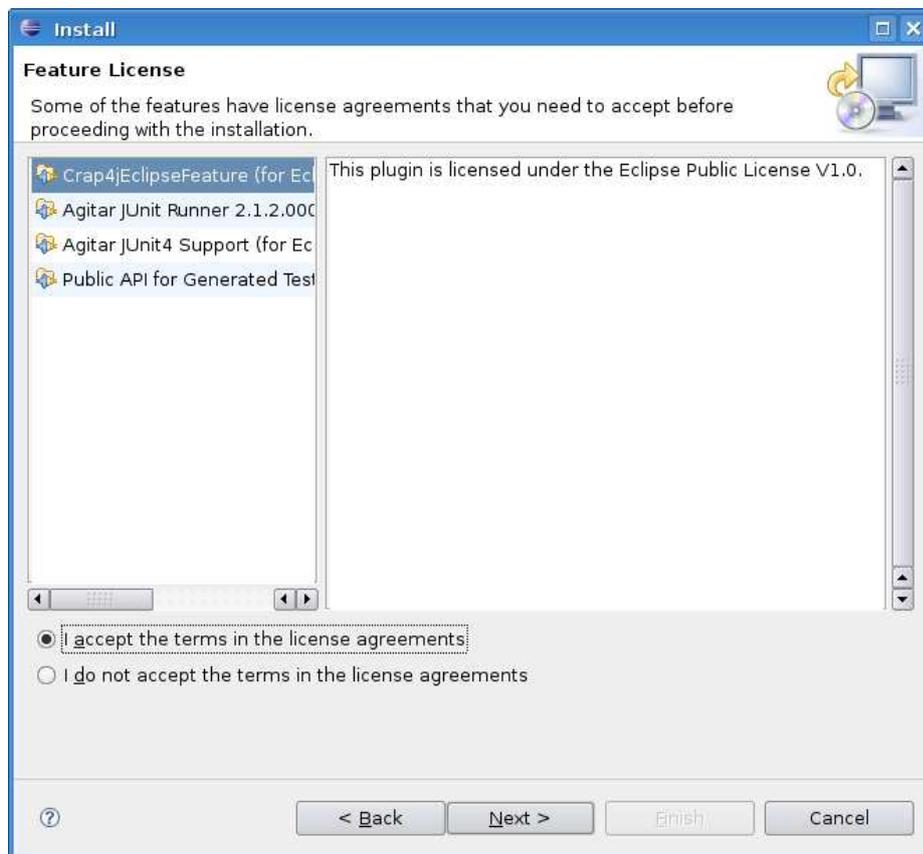
Y una vez ahí damos de alta un nuevo repositorio, lo llamamos `crap4j` e indicamos que la URL donde se ubica es <http://www.crap4j.org/downloads/update>



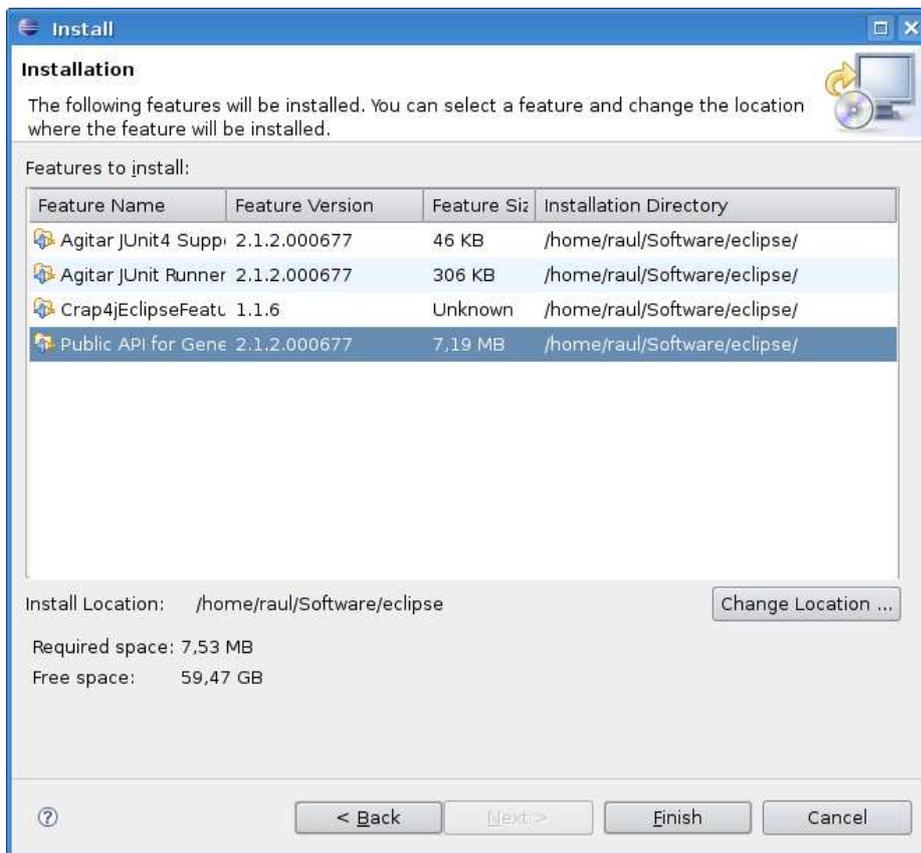
Una vez hayamos dado de alta el repositorio nos aparecerá una pantalla como la anterior, seleccionamos solo el repositorio `crap4j` y pulsamos 'Finish'. Tras eso nos aparecerá una ventana como la siguiente. Marcamos las opciones indicadas en la captura de pantalla y pulsamos sobre 'Next >'



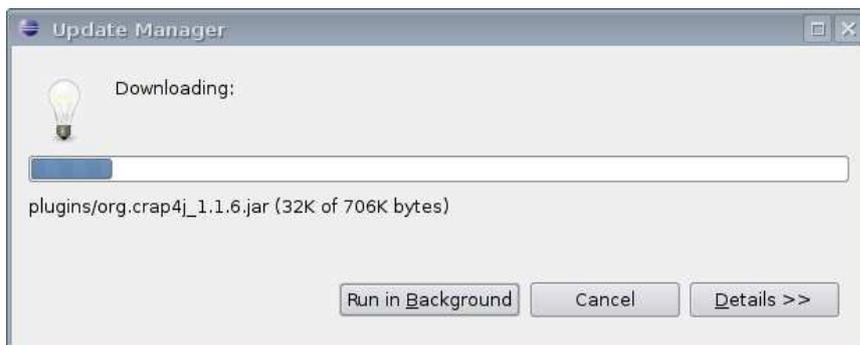
En caso de estar conforme con las licencias las aceptamos y seguimos



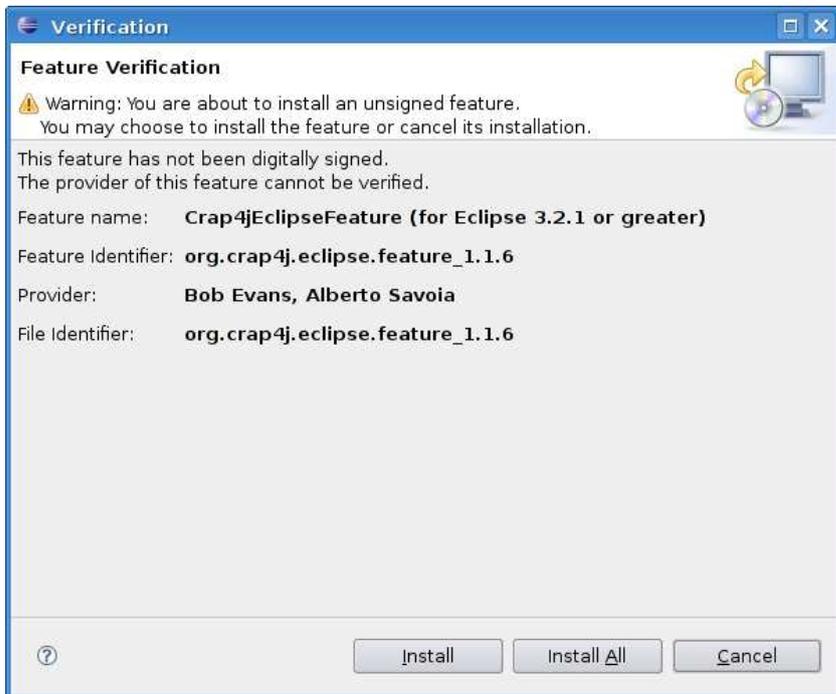
Nos avisará del lugar donde va a instalar el plugin por si queremos poner otra ruta. Pulsamos 'Finish'



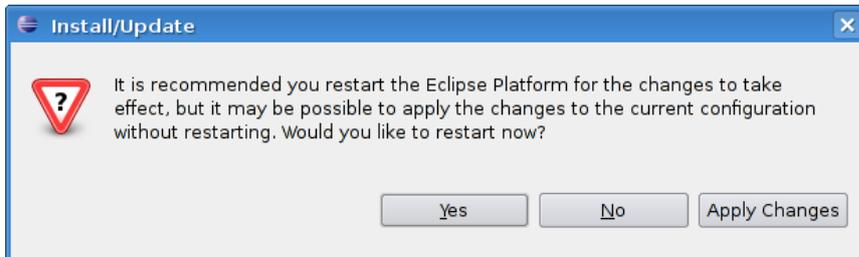
Esperamos a que se instale el plugin. Tardará más o menos en función al ancho de banda del que dispongamos.



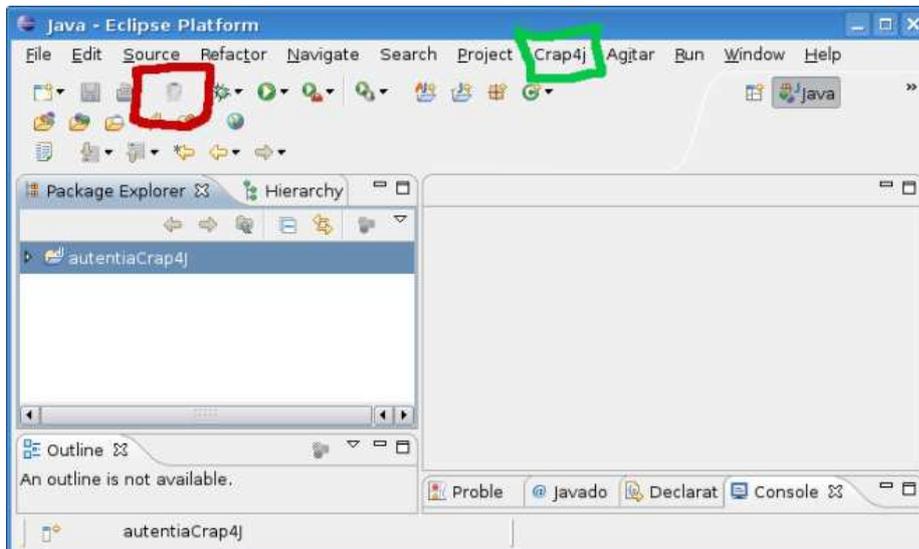
Una vez descargado lo instalamos con 'Install All' para no tener que ir aceptando los jar del plugin uno por uno.



Y tras eso os aconsejo reiniciar Eclipse para que se apliquen los cambios correctamente y así no tener problemas.



Al reiniciar veremos que el plugin se ha instalado correctamente si aparecen los elementos señalados en rojo y en verde. También aparecerá el submenú 'Agitar' a la derecha del submenú 'Crap4j', y es que **Agitar Software** es la empresa que está tras el desarrollo de Crap4j.



4. Funcionamiento del plugin

Para ejecutar el plugin basta con seleccionar un proyecto cualquiera y pulsar el botón remarcado en rojo en la imagen anterior. Tras la ejecución, en el directorio donde tengamos el proyecto, se nos creará un directorio llamado 'agitar/reports/crap4j' con un informe sobre lo que ha encontrado. Si no lo veis, usad el navegador de archivos o recargad el proyecto seleccionándolo y pulsando F5.

Voy a poner 3 ejemplos con los 3 resultados para que los interpretemos. Sobre un proyecto vacío creo la clase siguiente:

```
view plain print ?
01. package com.autentia.crap4j;
02.
03. public class CrappyClass {
04.
05.     public void CrappyMethod (final boolean param1, final boolean param2, final boolean param3) {
06.
07.         if (param1)
08.         {
09.             System.out.println ("param1 es true");
10.             if (param2)
11.             {
12.                 System.out.println ("param2 es true");
13.                 if (param3)
14.                 {
15.                     System.out.println ("param3 es true");
16.                 }
17.                 else
18.                 {
19.                     System.out.println ("param3 es false");
20.                 }
21.             }
22.             else
23.             {
24.                 System.out.println ("param2 es false");
25.                 if (param3)
26.                 {
27.                     System.out.println ("param3 es true");
28.                 }
29.                 else
30.                 {
31.                     System.out.println ("param3 es false");
32.                 }
33.             }
34.         }
35.     }
36.     else
37.     {
38.         System.out.println ("param1 es false");
39.         if (param2)
40.         {
41.             System.out.println ("param2 es true");
42.             if (param3)
43.             {
44.                 System.out.println ("param3 es true");
45.             }
46.             else
47.             {
48.                 System.out.println ("param3 es false");
49.             }
50.         }
51.         else
52.         {
53.             System.out.println ("param2 es false");
54.             if (param3)
55.             {
56.                 System.out.println ("param3 es true");
57.             }
58.             else
59.             {
60.                 System.out.println ("param3 es false");
61.             }
62.         }
63.     }
64. }
65. }
66. }
```

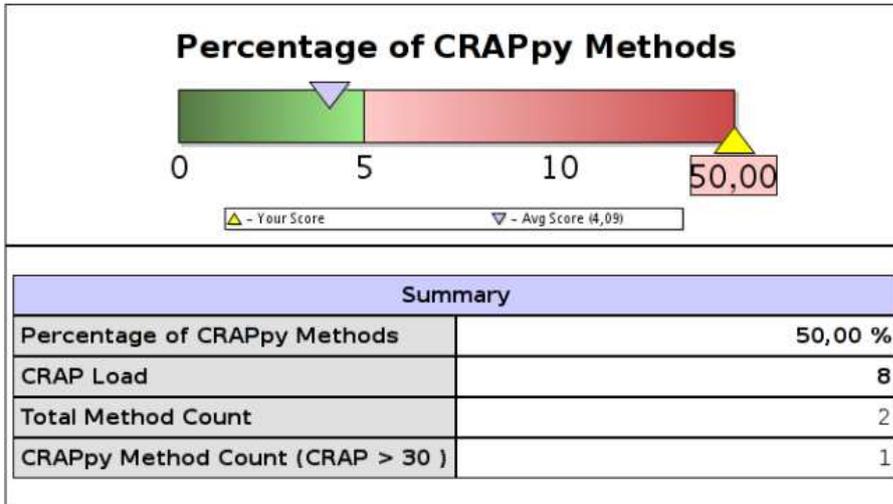
El funcionamiento de este método es muy simple: recibe 3 valores booleanos y muestra si son `true` o `false`. Sin embargo, si veis el código veis que es bastante malo porque tiene una complejidad ciclomática bastante elevada (muchos `if ... else` encadenados) y no tenemos pruebas unitarias para este método, con lo cual ante no podemos saber si cambiando código estamos estropeando otra parte de la aplicación. En este caso un cambio no afecta a nada, pero imagináros que estuviésemos abriendo o cerrando transacciones, cambiando el estado de alguna parte de la aplicación, definiendo navegabilidad, etc.

Ejecutamos `crap4j` pulsando sobre el botón recuadrado en rojo y vemos los resultados:

CRAP Report

Project: /home/raul/workspace/autentiaCrap4j

Generated at 20/01/08 14:07



El 50% de los métodos de la clase son CRAPpy (difícilmente mantenibles). Pero, ¿por qué dos si sólo hemos escrito un método?. Son dos porque está evaluando el constructor (en este caso el constructor por defecto) y el método `CrappyMethod()` que programamos anteriormente. Si vemos la mantenibilidad por métodos, veremos lo siguiente:

CRAP Report Detail

(Sorted by Crap Load)

Project: /home/raul/workspace/autentiaCrap4j

[Overview Page](#) | [CRAP](#) | [Complexity](#) | [Coverage](#)

Method	Complexity	Coverage	CRAP	CRAP Load
<code>public void CrappyMethod(boolean, boolean, boolean)</code> <small>com.autentia.crap4j.CrappyClass</small>	8	0,00 %	72,00	8
<code>public void <init>()</code> <small>com.autentia.crap4j.CrappyClass</small>	1	0,00 %	2,00	0

El código está cubierto al 0% por pruebas unitarias, la complejidad ciclomática del método `CrappyMethod()` es de 8 y su índice CRAP de 72. Un método se considera difícilmente mantenible cuando su índice CRAP es mayor que 30.

¿Cómo se puede arreglar esto?. Intentemos al menos crear una prueba unitaria que nos garantice que si cambiamos código los cambios no van a ocasionar ningún desastre. Vamos a crear una prueba unitaria que pase por varios `if ... else` para aumentar la cobertura de las pruebas unitarias sobre el código (lo cual no es muy difícil ya que ahora mismo es del 0,00%). La prueba que voy a crear siempre válida al método, con lo cual no es una prueba apropiada, pero estamos en un tutorial y todo es a modo didáctico. El código de la prueba es el siguiente:

```

view plain print ?
01. package com.autentia.crap4j;
02.
03. import org.junit.Assert;
04. import org.junit.Test;
05.
06. public class CrappyClassTest {
07.
08.     @Test
09.     public void testCrappyMethod() {
10.         CrappyClass crappyClass = new CrappyClass();
11.         crappyClass.CrappyMethod(true, false, false);
12.         crappyClass.CrappyMethod(false, true, true);
13.         Assert.assertTrue(true);
14.     }
15.
16. }

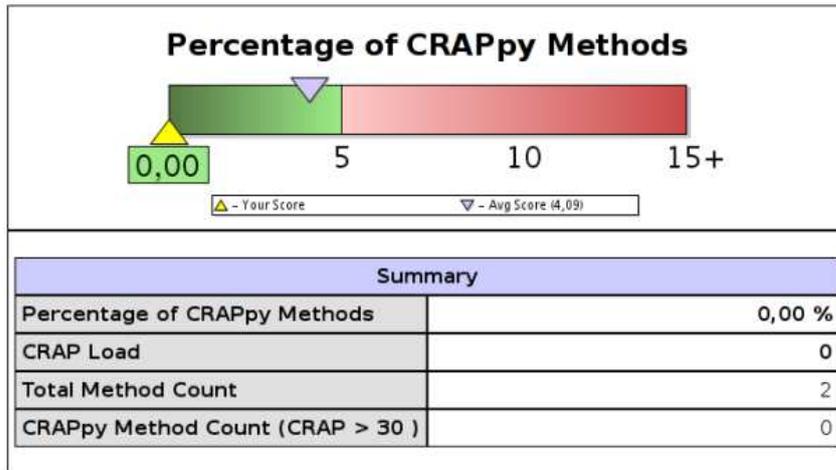
```

Hacemos click con el botón derecho sobre la clase 'CrappyClass.java', seleccionamos 'Agitar > Find Test for CrappyClass' y una vez ahí escogemos `CrappyClassTest`. Con esto lo que vamos a conseguir es que al lanzar `Crap4j` se van a lanzar las pruebas unitarias que hayamos indicado y, de este modo, va a ser posible saber qué grado de cobertura tienen las pruebas sobre el código. Lanzamos `Crap4j` y vemos los resultados:

CRAP Report

Project: /home/raul/workspace/autentiaCrap4j

Generated at 20/01/08 14:27



Vaya, ahora no tenemos métodos con un índice CRAP mayor que 30. ¿Qué índices tienen ahora los métodos?

CRAP Report Detail

(Sorted by Crap Load)

Project: /home/raul/workspace/autentiaCrap4j

[Overview Page](#) | [CRAP](#) | [Complexity](#) | [Coverage](#)

Method	Complexity	Coverage	CRAP	CRAP Load
public void <init>() com.autentia.crap4j.Crappy Class	1	100,00 %	1,00	0
public void CrappyMethod(boolean, boolean, boolean) com.autentia.crap4j.Crappy Class	8	44,83 %	18,75	0

El constructor ahora está cubierto al 100% por las pruebas y el método `CrappyMethod()` ha bajado de un CRAP de 72 a uno de 18.75 gracias a la cobertura de las pruebas unitarias, que alcanza un 44,83% del código del método. El índice es inferior a 30 así que se considera que no es difícil de mantener.

¿Y si creamos un método igual, solo que sin pruebas y con una complejidad ciclomática inferior?

```

view plain print ?
01. package com.autentia.crap4j;
02.
03. public class NonCrappyClass {
04.
05.     public void NonCrappyMethod (final boolean param1, final boolean param2, final boolean param3) {
06.         showContent ("param1", param1);
07.         showContent ("param2", param1);
08.         showContent ("param3", param1);
09.     }
10.
11.     private void showContent (final String name, final boolean value) {
12.
13.         if (value == true)
14.         {
15.             System.out.println (name + "es true");
16.         }
17.         else
18.         {
19.             System.out.println (name + "es false");
20.         }
21.     }
22. }

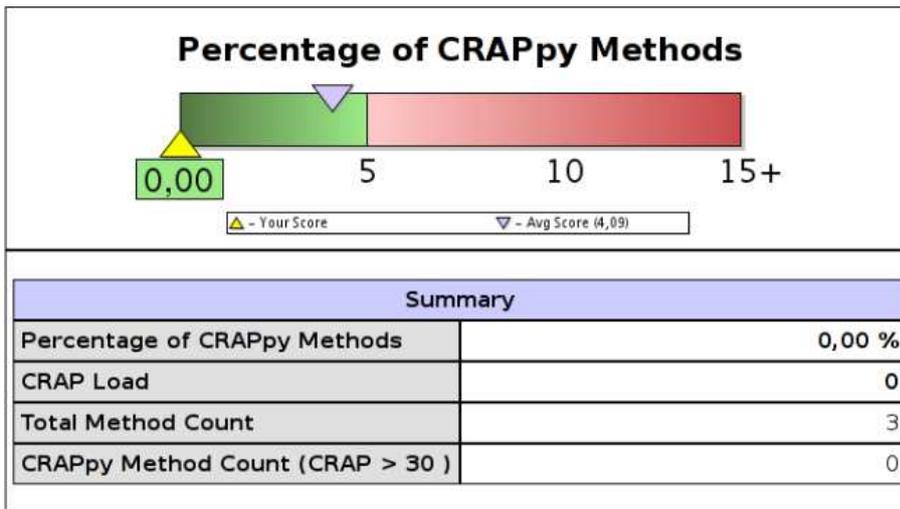
```

Habríamos obtenido el siguiente resultado:

CRAP Report

Project: /home/raul/workspace/autentiaCrap4j

Generated at 20/01/08 14:35



Ningún método con índice CRAP mayor a 30, y sin pruebas unitarias

CRAP Report Detail

(Sorted by Crap Load)

Project: /home/raul/workspace/autentiaCrap4j

[Overview Page](#) | [CRAP](#) | [Complexity](#) | [Coverage](#)

Method	Complexity	Coverage	CRAP	CRAP Load
public void <init>() com.autentia.crap4j.NonCrappyClass	1	0,00 %	2,00	0
public void NonCrappyMethod(boolean, boolean, boolean) com.autentia.crap4j.NonCrappyClass	1	0,00 %	2,00	0
private void showContent(java.lang.String, boolean) com.autentia.crap4j.NonCrappyClass	2	0,00 %	6,00	0

Y encima el índice CRAP mayor es de 6, y porque no hay pruebas unitarias, porque el método `showContent()` es muy fácilmente mantenible (aunque faltarían comprobaciones ya que, por ejemplo, el parámetro `name` podría recibirse con un valor `null`).

5. Conclusiones

Pues como podeis ver el índice CRAP es bastante útil y puede ayudar bastante sobre todo a la hora de iniciar un proyecto, ya que gracias a él nos aseguraremos que, desde un principio, nuestra aplicación va a ser fácilmente mantenible mientras la desarrollemos o, al menos, vamos a poder tener localizados los puntos donde este mantenimiento no va a ser fácil en un futuro si éste es necesario.

Como no podría ser de otro modo, también va a ayudarnos a saber si es posible realizar una refactorización de un modo fácil una vez llegado el momento. Esto puede ser muy útil si estamos siguiendo una metodología de desarrollo ágil.

Para terminar, simplemente indicar que el índice CRAP, y por ende el plugin, lo que hace es "obligarnos" a seguir unas buenas prácticas de programación que nos haga la vida más sencilla a medio y largo plazo, y seguramente a corto plazo también, como son:

- hacer los métodos lo más sencillos y cortos posibles.
- usar pruebas unitarias.

Como dice un amigo mio "hay que intentar que el pan para hoy no sea hambre para mañana". El índice CRAP lo que nos indica es hasta qué punto podemos pasar hambre mañana :-)

Espero que os sea de utilidad.

- Puedes opinar sobre este tutorial [haciendo clic aquí](#).
- Puedes firmar en nuestro libro de visitas [haciendo clic aquí](#).
- Puedes asociarte al grupo AdictosAlTrabajo en XING [haciendo clic aquí](#).

- Añadir a favoritos Technorati.  **ADD THIS BLOG TO MY FAVORITES**



Esta obra está licenciada bajo [licencia Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5](#)

Recuerda

Autentia te regala la mayoría del conocimiento aquí compartido ([Ver todos los tutoriales](#)). Somos expertos en: J2EE, Struts, JSF, C++, OOP, UML, UP, Patrones de diseño ... y muchas otras cosas.

¿Nos vas a tener en cuenta cuando necesites consultoría o formación en

tu empresa?, ¿Vas a ser tan generoso con nosotros como lo tratamos de ser con vosotros?

Somos pocos, somos buenos, estamos motivados y nos gusta lo que hacemos ...

Autentia = Soporte a Desarrollo & Formación.

info@autentia.com

{ Block }

Servicio de notificaciones:

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales.

Formulario de suscripción a novedades:

E-mail

Tutoriales recomendados

Nombre	Resumen	Fecha	Visitas	pdf
PMD, Eclipse y NetBeans	Tutorial que describe la instalación y uso de PMD en los entornos de desarrollo Eclipse y NetBeans	2006-05-24	6146	pdf
Patrones de diseño J2EE	Os mostramos una interpretación particular de los patrones de diseño J2EE	2004-03-23	34060	pdf
Optimizando código Java con Eclipse Test Performance Tools Platform	En este tutorial vamos a aprender como usar Eclipse Test Performance Tools Platform (TPTP), que nos permite analizar nuestro código	2006-11-21	3140	pdf
Mejora de la calidad del código fuente con Eclipse	En este pequeño tutorial se muestra una de esas multiples opciones de la plataforma de desarrollo Eclipse que muchos de nosotros no vemos y que nos hubieran ahorrado un gran esfuerzo si lo hubieramos conocido en su día.	2007-02-26	2852	pdf
Pruebas Web con JWebUnit	Os mostramos como automatizar las pruebas de caja negra (desde el punto de vista de usuario final) de vuestro Web con el Framework gratuito JWebUnit. Esta técnica es perfecta para crear test de regresión de aplicaciones Web complejas.	2004-06-30	8149	pdf
SpringIDE, plugin de Spring para Eclipse	En adictosaltrabajo os hemos ido presentando diversos plugins para Eclipse. Esta vez le toca el turno a SpringIDE, un plugin que os ayudará a desarrollar aplicaciones que utilicen Spring.	2008-01-19	50	pdf
Aplicación de Patrones de Diseño en Java	En este tutorial os mostramos como las técnicas avanzadas de diseño (como patrones de diseño) contribuyen a la contrucción de aplicaciones profesionales en Java.	2004-05-17	29876	pdf
JUnit 4. Pruebas de Software Java	Tutorial que describe como utilizar la herramienta JUnit 4 para realizar pruebas de integridad y errores sobre Java.	2006-06-02	7908	pdf
Optimización Java con Eclipse Profiler Plugin	Alejandro Pérez nos enseña como analizar el rendimiento de nuestras aplicaciones con Eclipse Profiler Plugin.	2004-07-21	17179	pdf
Test con JUnit	Cuando se hacen desarrollo profesionales, no basta con hacer los programas, hay que asegurarse de que van a funcionar. Una de las técnicas más seguras es crear aplicaciones que incluyan el código para autoprobarse. Os mostramos como usar JUnit	2003-06-21	16451	pdf

Nota:

Los tutoriales mostrados en este Web tienen como objetivo la difusión del conocimiento. Los contenidos y comentarios de los tutoriales son responsabilidad de sus respectivos autores. En algún caso se puede hacer referencia a marcas o nombres cuya propiedad y derechos es de sus respectivos dueños. Si algún afectado desea que incorporemos alguna reseña específica, no tiene más que solicitarlo. Si alguien encuentra algún problema con la información publicada en este Web, rogamos que informe al administrador rcanales@adictosaltrabajo.com para su resolución.