UNIVERSIDAD NACIONAL DE UCAYALI
VICERRECTORADO DE INVESTIGACIÓN
DIRECCIÓN GENERAL DE PRODUCCIÓN INTELECTUAL
8/8/17
OFICINA EJECUTIVA DE BIBLIOTECA Y PUBLICACIONES

UNIVERSIDAD NACIONAL DE UCAYALI

FACULTAD DE INGENIERÍA DE SISTEMAS Y DE INGENIERÍA CIVIL

ESCUELA DE FORMACIÓN PROFESIONAL DE INGENIERÍA DE SISTEMAS

"SONARQUBE 4.0 EN LA CERTIFICACIÓN DE CÓDIGO JAVA EN UNA CONSULTORA DE PROYECTOS INFORMÁTICOS"

Tesis

Para Optar al Título de:

INGENIERO DE SISTEMAS

TESISTA : **BACH. AGUILAR POLO YOSLAR.**

BACH. FLORES CHIRINOS GERARD.

ASESOR : **ING. MG. JORGE LUIS HILARIO RIVAS.**

PUCALLPA – PERU

2017

UNIVERSIDAD NACIONAL DE UCAYALI

FACULTAD DE INGENIERÍA DE SISTEMAS Y DE INGENIERIA CIVIL

Escuela Académica Profesional de Ingeniería de Sistemas

INFORME DE ASESORÍA DE TESIS

I. Tesista : Bach. Aguilar Polo Yoslar.

Bach. Flores Chirinos Gerard.

II. Tesis : SONARQUBE 4.0 EN LA CERTIFICACIÓN DE CÓDIGO JAVA EN UNA CONSULTORA DE PROYECTOS INFORMÁTICOS.

Que, los tesistas han cumplido con ejecutar la tesis titulada: "SONARQUBE 4.0 EN LA CERTIFICACIÓN DE CÓDIGO JAVA EN UNA CONSULTORA DE PROYECTOS INFORMÁTICOS", de conformidad con el Reglamento General de Grados y Títulos de la Universidad Nacional de Ucayali, por lo que mi asesoría declara: **APROBADO** y, encontrándose apta para ser presentada y evaluada por la Comisión de Grados y Títulos de la Facultad de Ingeniería de Sistemas y de Ingeniería Civil de la Universidad Nacional de Ucayali.

Se expide el presente documento, a solicitud de los interesados para los fines consiguientes.

Pucallpa, de del 2017

Ing. Mg. Jorge Luis Hilario Rivas.

Asesor de Tesis

"SONARQUBE 4.0 EN LA CERTIFICACIÓN DE CÓDIGO JAVA EN UNA CONSULTORA DE PROYECTOS INFORMÁTICOS"

Por:

Bach. Aguilar Polo Yoslar.

Bach. Flores Chirinos Gerard.

Aprobado

Jurado 1.

Jurado 2.

Jurado 3.

Asesor:

Ing. Mg. Jorge Luis Hilario Rivas.

DEDICATORIA

A mis padres y familiares.

Yoslar.

A ms padres, a mi esposa y mi hija.

Gerard.

AGRADECIMIENTO

Al Ing. Mg. Jorge Luis Hilario Rivas, por su asesoramiento en nuestro informe, nuestro mayor reconocimiento por su trayectoria académica y profundo respeto, le estamos muy agradecidos.

A la Universidad Nacional de Ucayali y los catedráticos de la Escuela Profesional de Ingeniería de sistemas.

ÍNDICE DE CONTENIDO

DEDICATORIA iv

AGRADECIMIENTO v

LISTA DE TABLAS xii

LISTA DE FIGURAS xvi

RESUMEN xix

ABSTRACT xx

INTRODUCCION xxi

CAPITULO I 1

PLANTEAMIENTO DEL PROBLEMA 1

1.1. DESCRIPCIÓN DE LA REALIDAD PROBLEMÁTICA 1

1.2. FORMULACIÓN DEL PROBLEMA 2

1.2.1. GENERAL 2

1.2.2. ESPECÍFICOS 3

1.3. OBJETIVOS DE LA INVESTIGACIÓN 3

1.3.1. GENERAL 3

1.3.2. ESPECÍFICOS 3

1.4. HIPÓTESIS 4

1.4.1. GENERAL 4

1.4.2. ESPECÍFICOS 4

1.5. JUSTIFICACIÓN, IMPORTANCIA Y LIMITACIONES DE LA INVESTIGACIÓN 5

1.5.1. JUSTIFICACIÓN TEÓRICA 5

1.5.2. JUSTIFICACIÓN PRÁCTICA 5

1.5.3. JUSTIFICACIÓN METODOLÓGICA 6

1.5.4. IMPORTANCIA 6

1.5.5. LIMITACIONES 7

1.6. VARIABLES. 7

1.6.1. VARIABLE INDEPENDIENTE 7

1.6.2. VARIABLE DEPENDIENTE 7

1.6.3. VARIABLE INTERVINIENTE 7

1.7. OPERACIONALIZACIÓN DE LAS VARIABLES 8

CAPITULO II 9

MARCO TEÓRICO 9

2.1. ANTECEDENTES DE LA INVESTIGACIÓN 9

2.2. BASES TEÓRICAS 12

JAVA 12

CARACTERÍSTICAS DE JAVA COMO LENGUAJE DE PROGRAMACIÓN 14

CODIGO FUENTE 16

SONARQUBE 18

ARQUITECTURA DE SONARQUBE 19

INTEGRACIÓN DE SONARQUBE 21

REQUISITOS DE HARDWARE PARA SONARQUBE 22

MEDIDA DE LA CALIDAD DE TU CÓDIGO 23

JENKINS 23

2.3. DEFINICIÓN DE TÉRMINOS BÁSICOS 25

CAPITULO III 29

MARCO METODOLÓGICO 29

3.1. METODOLOGÍA DE LA INVESTIGACIÓN 29

3.1.1. TIPO DE INVESTIGACIÓN 29

3.1.2. NIVEL DE INVESTIGACIÓN 29

3.1.3. MÉTODO DE INVESTIGACIÓN 29

3.1.4. DISEÑO DE INVESTIGACIÓN 30

3.2. POBLACIÓN Y MUESTRA 31

3.2.1. POBLACIÓN. 31

3.2.2. MUESTRA 31

3.3. PROCESAMIENTO DE RECOLECCIÓN DE DATOS. 32

3.4. TRATAMIENTO DE LOS DATOS 32

3.4.1. PROCESAMIENTO Y PRESENTACIÓN DE DATOS 32

3.4.2. MECANISMOS PARA EL PROCESAMIENTO Y PRESENTACIÓN DE DATOS 33

3.4.3. MECANISMOS PARA EL ANÁLISIS E INTERPRETACIÓN DE RESULTADOS 34

CAPITULO IV 35

RESULTADOS 35

4.1. INTERPRETACIÓN DE RESULTADOS PRE – TEST 35

4.2. INTERPRETACIÓN DE RESULTADOS POST – TEST 46

4.3. PRUEBA DE HIPÓTESIS 57

4.3.1. FORMULACIÓN DE CUADRO DE VALORES DE INDICADORES 57

4.3.2. SUPUESTOS DE LA PRUEBA DE HIPÓTESIS 58

4.3.3. CÁLCULO DEL VALOR CRÍTICO Y LA FUNCIÓN PRUEBA 59

CAPITULO V 64

DISCUSIÓN DE RESULTADOS 64

5.1. CONSTRASTACIÓN DE RESULTADOS 64

5.1.1. DEFINICIÓN DE MÉTRICAS 66

5.1.2. DEFINICIÓN DE MÉTRICAS JAVA 67

5.1.3. ANÁLISIS DE CODIGO FUENTE 76

5.1.4. EL ESPACIO DE TRABAJO 100

5.1.5. LA PÁGINA DE INICIO 102

5.1.6. CUADRO DE MANDOS DE UN PROYECTO 111

5.1.7. NAVEGACIÓN POR LAS MÉTRICAS 116

5.1.8. EVENTOS 117

5.1.9. EL VISOR DE RECURSOS 117

5.1.10. COMPONENTES 125

5.1.11. NAVEGACIÓN POR LAS EVIDENCIAS 126

5.1.12. LA MÁQUINA DEL TIEMPO 128

5.1.13. NUBES 132

5.1.14. LOS "TOP" 133

5.1.15. DISEÑO Y DEPENDENCIAS 134

5.1.16. LIBRERÍAS 138

5.1.17. SEGURIDAD 140

5.1.18. REVISIONES MANUALES DE CÓDIGO 142

5.1.19. USANDO SONAR EN ECLIPSE 146

CONCLUSIONES 160

SUGERENCIAS 161

REFERENCIAS BIBLIOGRÁFICAS 162

ANEXOS 165

ANEXO 1. MATRIZ DE IDENTIFICACIÓN DEL PROBLEMA 166

ANEXO 2. HOJA DE RUTA DE LAS VARIABLES 167

ANEXO 3. MATRIZ DE CONSISTENCIA 168

ANEXO 4. CUESTIONARIO DIRIGIDO AL EQUIPO QUE ELABORAN PROYECTOS DE LA CONSULTORA EVERIS 169

ANEXO 5. VALIDACIÓN DEL INSTRUMENTO 171

ANEXO 6. MANUAL DE USUARIO JENKINS 173

ANEXO 7. MANUAL DE USUARIO SONARQUBE 184

LISTA DE TABLAS

Tabla 1. Dimensiones e indicadores de las variables 8

Tabla 2. Población del equipo de software..... 31

Tabla 3. Instrumentos de recolección de datos..... 32

Tabla 4. Pre – Test ¿Cómo califica la funcionalidad del código Java?..... 35

Tabla 5. Pre – Test ¿Cómo califica la confiabilidad del código Java? 36

Tabla 6. Pre – Test ¿Cómo califica la usabilidad del código Java?..... 37

Tabla 7. Pre – Test ¿Cómo califica la eficiencia del código Java? 38

Tabla 8. Pre – Test ¿Cómo califica la eficacia del código Java?..... 39

Tabla 9. Pre – Test ¿Cómo califica la portabilidad del código Java?..... 40

Tabla 10. Pre – Test ¿Cómo califica los reportes emitidos en la certificación de código Java?..... 41

Tabla 11. Pre – Test ¿Cómo califica la verificación en la certificación de código Java? 42

Tabla 12. Pre – Test ¿Cómo califica los procesos evaluación en la certificación de código Java?..... 43

Tabla 13. Pre – Test ¿Cómo califica los recursos en la certificación de código Java? 44

Tabla 14. Pre – Test ¿Cómo califica la satisfacción en la certificación de código Java? 45

Tabla 15. ¿Cómo califica la funcionalidad del SonarQube 4.0 en la certificación de código Java? 46

Tabla 16. ¿Cómo califica la confiabilidad del SonarQube 4.0 en la certificación de código Java? 47

Tabla 17. ¿Cómo califica la usabilidad del SonarQube 4.0 en la certificación de código Java? 48

Tabla 18. ¿Cómo califica la eficiencia del SonarQube 4.0 en la certificación de código Java? 49

Tabla 19. ¿Cómo califica la eficacia del SonarQube 4.0 en la certificación de código Java? 50

Tabla 20. ¿Cómo califica la portabilidad del SonarQube 4.0 en la certificación de código Java? 51

Tabla 21. ¿Cómo califica los reportes emitidos del SonarQube 4.0 en la certificación de código Java? 52

Tabla 22. ¿Cómo califica la verificación del SonarQube 4.0 en la certificación de código Java? 53

Tabla 23. ¿Cómo califica los procesos del SonarQube 4.0 en la certificación de código Java? 54

Tabla 24. ¿Cómo califica los recursos del SonarQube 4.0 en la certificación de código Java? 55

Tabla 25. ¿Cómo califica la satisfacción del SonarQube 4.0 en la certificación de código Java? 56

Tabla 26. Para variables cualitativos: 58

Tabla 27. Resumen de resultado de valores 59

Tabla 28. Correlación de datos del Pre – Test 62

Tabla 29. Correlación de datos del Post – Test 63

Tabla 30. Arquitectura del Sonar 64

Tabla 31. Calidad del Sonar 65

Tabla 32. Interface web del Sonar 66

Tabla 33. Definición de métricas Java por tamaño 67

Tabla 34. Definición de métricas Java por pruebas 70

Tabla 35. Definición de métricas Java por duplicidad 72

Tabla 36. Definición de métricas Java por diseño 72

Tabla 37. Definición de métricas Java por complejidad 73

Tabla 38. Definición de métricas Java por reglas 75

Tabla 39. Definición de métricas Java por SCM 75

Tabla 40. Servidor Sonar 95

Tabla 41. Base de datos 95

Tabla 42. Configuración del proyecto 95

Tabla 43. Configuración de proyecto 96

Tabla 44. Unit And Integration Tests 98

Tabla 45. Sonar Configuration 99

Tabla 46. Log 99

LISTA DE FIGURAS

Figura 1. Las interioridades de Java 14

Figura 2. Ejes de calidad 19

Figura 3. Arquitectura de SonarQube 20

Figura 4. Integración de SonarQube 22

Figura 5. Pre – Test ¿Cómo califica la funcionalidad del código Java? 35

Figura 6. Pre – Test ¿Cómo califica la confiabilidad del código Java? 36

Figura 7. Pre – Test ¿Cómo califica la usabilidad del código Java? 37

Figura 8. Pre – Test ¿Cómo califica la eficiencia del código Java? 38

Figura 9. Pre – Test ¿Cómo califica la eficacia del código Java? 39

Figura 10. Pre – Test ¿Cómo califica la portabilidad del código Java? 40

Figura 11. Pre – Test ¿Cómo califica los reportes emitidos en la certificación de código Java? 41

Figura 12. Pre – Test ¿Cómo califica la verificación en la certificación de código Java? 42

Figura 13. Pre – Test ¿Cómo califica los procesos evaluación en la certificación de código Java? 43

Figura 14. Pre – Test ¿Cómo califica los recursos en la certificación de código Java? 44

Figura 15. Pre – Test ¿Cómo califica la satisfacción en la certificación de código Java? 45

Figura 16. ¿Cómo califica la funcionalidad del SonarQube 4.0 en la certificación de código Java? 46

Figura 17. ¿Cómo califica la confiabilidad del SonarQube 4.0 en la certificación de código Java? 47

Figura 18. ¿Cómo califica la usabilidad del SonarQube 4.0 en la certificación de código Java? 48

Figura 19. ¿Cómo califica la eficiencia del SonarQube 4.0 en la certificación de código Java? 49

Figura 20. ¿Cómo califica la eficacia del SonarQube 4.0 en la certificación de código Java? 50

Figura 21. ¿Cómo califica la portabilidad del SonarQube 4.0 en la certificación de código Java? 51

Figura 22. ¿Cómo califica los reportes emitidos del SonarQube 4.0 en la certificación de código Java? 52

Figura 23. ¿Cómo califica la verificación del SonarQube 4.0 en la certificación de código Java? 53

Figura 24. ¿Cómo califica los procesos del SonarQube 4.0 en la certificación de código Java? 54

Figura 25. ¿Cómo califica los recursos del SonarQube 4.0 en la certificación de código Java? 55

Figura 26. ¿Cómo califica la satisfacción del SonarQube 4.0 en la certificación de código Java? 56

RESUMEN

La presente investigación, cuenta con el problema general de qué manera SonarQube 4.0 mejora la certificación de código Java en una consultora de proyectos informáticos. El objetivo consiste en proponer el SonarQube 4.0 mejora la certificación de código Java en una consultora de proyectos informáticos. La hipótesis demostrada SonarQube 4.0 mejora la certificación de código Java en una consultora de proyectos informáticos.

El tipo y nivel de la investigación es aplicada y descriptiva respectivamente, asimismo el diseño es Pre Test – Post; la muestra corresponde a 16 profesionales del equipo que elaboran proyectos de software de la consultora Everis, y el instrumento que sirvió para recolectar información es la encuesta.

La principal conclusión fue se ejecuto un estudio detallado para la respectiva certificación de codigo Java, donde se detectan vulnerabililidades en las aplicaciones webs que deben ser corregidas. En primera instancia se realiza una revisión general del codigo fuente, asimismo se describe el uso de la técnica para ejecutar el respectivo análisis, en cumplimiento de los estándares de calidad.

Palabras claves: Proceso, Metodología y Usuario.

ABSTRACT

This research has the general problem of how SonarQube 4.0 improves Java code certification in a computer project consultancy. The objective is to propose the SonarQube 4.0 enhances Java code certification in a computer projects consultancy. The proven SonarQube 4.0 assumption improves Java code certification in a computer project consultancy.

The type and level of the research is applied and descriptive respectively, also the design is Pre Test - Post; The sample corresponds to 16 professionals of the equipment that elaborate software projects of the Everis consultancy, and the instrument that served to gather information is the survey.

The main conclusion was that a detailed study was carried out for the respective Java code certification, where vulnerabilities are detected in web applications that need to be corrected. In the first instance, a general revision of the source code is performed, as well as the use of the technique to execute the respective analysis, in compliance with the quality standards.

Key words: Process, Methodology and User.

INTRODUCCION

La tesis denominada "SONARQUBE 4.0 EN LA CERTIFICACIÓN DE CÓDIGO JAVA EN UNA CONSULTORA DE PROYECTOS INFORMÁTICOS", propone una revisión general del estado de las herramientas de análisis de código enfocadas en detectar vulnerabilidades en aplicaciones web y su implementación dentro del ciclo de desarrollo del software. Estas herramientas constituyen una forma rápida, eficiente y sencilla de asegurar los proyectos de software, incluso si no todos los fallos son detectados.

La evaluación busca demostrar el estado actual y real de los programas de, con el fin que SonarQube 4.0 logre mejorar la seguridad de sus proyectos a partir de la implementación de la herramienta que mas se ajuste a sus necesidades y metodologías de desarrollo.

Después de llevar a cabo el trabajo de campo, el presente trabajo de investigación se ha estructurado en cinco capítulos:

- **Capitulo I – Planteamiento del Problema:** Se describe los antecedentes del problema, diagnóstico de la situación actual, formulación del problema, objetivos, justificación e importancia, hipótesis, variables y operacionalización.
- **Capitulo II – Marco Teórico:** Se describe los antecedes del problema teóricamente, las bases teóricas y la definición de términos básicos.
- **Capitulo III – Marco Metodológico:** se describe el tipo, nivel método y diseño de la investigación, población y muestra, procesamiento de recolección de datos y el tratamiento de los datos.

- **Capitulo IV – Resultados:** Se presentan los resultados del análisis estadístico y la prueba de hipótesis.
- **CAPITULO V – Discusión de resultados:** Se realiza la discusión de resultados y la propuesta de la herramienta.

Finalmente, presentamos las conclusiones, sugerencias, referencias bibliográficas y anexos correspondientes.

CAPITULO I

PLANTEAMIENTO DEL PROBLEMA

1.1. DESCRIPCIÓN DE LA REALIDAD PROBLEMÁTICA

En el Perú existen consultoras de proyectos informáticos, con un amplio portafolio de servicios y de clientes, el inconveniente radica que los proyectos ejecutados, no cuentan con una debida estandarización de los procesos para la certificación del codigo fuente, es por ello que la presenta investigación, plantea evaluar en la etapa de "Prueba" si los códigos mantienen uniformidad y pueden ser reusables en próximas versiones.

El análisis del codigo es un proceso automatizado que se realiza sobre la codificación de una aplicación sin necesidad de la ejecución. Puede incluirse el análisis estático manual realizado por un humano. Pero, para términos de esta tesis, el análisis estático es un análisis realizado de forma automatizada por una herramienta. La idea principal de estas herramientas es encontrar, tan pronto como sea posible, defectos de codificación en el ciclo de vida del desarrollo, generalmente después de escribir el código fuente de la aplicación.

Las primeras herramientas de análisis de código estático eran sencillos programas de chequeo que no eran mas sofisticados. Estos software fueron rápidamente precedidos por herramientas comerciales mas sofisticadas que buscaban automatizar la revisión de código para una variedad de lenguajes de programación. Estos programas analizan los

fallos mas comunes para cada lenguaje en particular y ayudan a los desarrolladores a crear códigos mas estables y de mayor calidad.

SÍNTOMAS

- Falta de normativas y protocolos.
- Conflictos al momento de realizar la siguiente versión del software.
- Las clases y funciones no se encuentran debidamente relacionadas.
- Clases con líneas de código extenso.

CONSECUENCIAS

- Duplicidad de codigo en las clases.
- Reestructuración total del código fuente en la siguiente versión del software.
- Uso excesivo de variables locales.
- Errores al momento de la compilación.

1.2. FORMULACIÓN DEL PROBLEMA

1.2.1. GENERAL

¿De qué manera SonarQube 4.0 mejora la certificación de código Java en una consultora de proyectos informáticos?

1.2.2. ESPECÍFICOS

1. ¿Cuál es el proceso de certificación de código Java en el ciclo de desarrollo de los proyectos informáticos para el cumplimiento de estándares de calidad?
2. ¿Cuál es el grado de asociación entre SonarQube 4.0 y la certificación de código Java en una consultora de proyectos informáticos?
3. ¿Cuál es la aplicación del SonarQube 4.0 como herramienta para la certificación de código Java en una consultora de proyectos informáticos?

1.3. OBJETIVOS DE LA INVESTIGACIÓN

1.3.1. GENERAL

Proponer el SonarQube 4.0 mejora la certificación de código Java en una consultora de proyectos informáticos.

1.3.2. ESPECÍFICOS

1. Integrar la certificación de código Java en el ciclo de desarrollo de los proyectos informáticos para el cumplimiento de estándares de calidad.
2. Determinar el grado de asociación entre SonarQube 4.0 y la certificación de código Java en una consultora de proyectos informáticos.

3. Evaluar la aplicación del SonarQube 4.0 como herramienta para la certificación de código Java en una consultora de proyectos informáticos.

1.4. HIPÓTESIS

1.4.1. GENERAL

"SonarQube 4.0 mejora la certificación de código Java en una consultora de proyectos informáticos".

1.4.2. ESPECÍFICOS

1. "Existe integración de la certificación de código Java en el ciclo de desarrollo de los proyectos informáticos para el cumplimiento de estándares de calidad".

2. "SonarQube 4.0 posee un grado de asociación con la certificación de código Java en una consultora de proyectos informáticos".

3. "Es viable la aplicación del SonarQube 4.0 como herramienta para la certificación de código Java en una consultora de proyectos informáticos".

1.5. JUSTIFICACIÓN, IMPORTANCIA Y LIMITACIONES DE LA INVESTIGACIÓN

1.5.1. JUSTIFICACIÓN TEÓRICA

Mediante la documentación fundamentamos nuestra investigación, porque en ella expresamos nuestras ideas que son respaldadas por autores con definiciones acertadas. De igual forma con el uso del SonarQube 4.0 lograremos importantes mejoras en la certificación del código Java, anulando código redundante en la elaboración de las clases.

El sistema brinda lo siguiente:

- Resumen de las incidencias.
- Reporte de los riesgos alto, medio y bajo.
- Cumplimiento de reglas.
- Reporte del codigo Java.

1.5.2. JUSTIFICACIÓN PRÁCTICA

La aplicación del SonarQube 4.0 constituye una eficiente y segura forma de confirmar, el adecuado uso de los proyectos informáticos, asimismo si los errores no son detectados. Mediante la evaluación buscamos señalar el estado real de los códigos fuente elaborados en el lenguaje de programación Java, con el propósito de mejorar

la seguridad en la implementación realizando ajustes y métodos de desarrollo.

1.5.3. JUSTIFICACIÓN METODOLÓGICA

Se siguió los lineamientos del proceso de investigación la cual incluye el planteamiento de interrogantes, objetivos e hipótesis, a fin de establecer un conocimiento probable acerca de los factores que influyen en el certificación del código Java (Hernández Sampieri, Fernández Collado, & Baptista Lucio, 2014).

1.5.4. IMPORTANCIA

La importancia del Sonarqube radica en la mejora continua del codigo fuente, manteniendo un cierto nivel de calidad y legibilidad de código, siendo de crucial importancia para un desarrollo exitoso en el ambiente de desarrollo dinámico actual de las consultoras informáticas.

Esos ambientes requieren seguir ciertas convenciones de código para lograr que el código sea entendible para todos los involucrados en el proceso.

Para mantener un control de alta calidad, elegimos proponer la revisión de código en nuestro proceso de desarrollo. Sin embargo, luego de cierto tiempo, las revisiones comenzaron a llevar mucho tiempo y desviaban nuestra atención de otras tareas importantes. La revisión de código consiste no sólo en un proceso de revisión

creativo sino también en un control de errores repetitivos, del uso de las reglas de convención, etcétera. Se cuestionó el proceso completo, así que una solución lógica era automatizar una parte.

1.5.5. LIMITACIONES

La disponibilidad de tiempo de los tesistas, por el término de la distancia entre ellos.

Falta de información en libros sobre la herramienta Sonarqube, teniendo que consideras bibliografías de la web.

Al estar laborando fuera de la ciudad de Pucallpa, los trámites tienden a demorar.

1.6. VARIABLES.

1.6.1. VARIABLE INDEPENDIENTE

SonarQube 4.0.

1.6.2. VARIABLE DEPENDIENTE

Certificación de codigo Java.

1.6.3. VARIABLE INTERVINIENTE

Consultora de proyectos informáticos.

1.7. OPERACIONALIZACIÓN DE LAS VARIABLES

Tabla 1. Dimensiones e indicadores de las variables

Variables	Dimensiones	Indicadores	
SonarQube 4.0.	Métricas de software	**1)** Funcionalidad	%
		2) Confiabilidad	%
		3) Usabilidad	%
		4) Eficiencia	%
		5) Eficacia	%
		6) Portabilidad	%
Certificación de codigo Java	Control de calidad	**1)** Reportes	%
		2) Verificación	%
		3) Procesos	%
		4) Recursos	%
		5) Satisfacción	%

CAPITULO II

MARCO TEÓRICO

2.1. ANTECEDENTES DE LA INVESTIGACIÓN

- (Ospina Delgado, 2015), en su investigación: *"Análisis de seguridad y calidad de aplicaciones (Sonarqube)"* busca Implantar dentro del ciclo de vida del desarrollo de aplicaciones el cómputo automático de una serie de métricas sobre la calidad del código y la búsqueda de vulnerabilidades. Lo anterior se hará realizando la integración de estos módulos de análisis dentro de la herramienta de software libre SonarQube. Llegando a la siguiente conclusión en el presente trabajo se realizó un estudio sistemático de la actualidad del análisis de código estático y, para ser más precisos, del análisis de código estático enfocado en detectar vulnerabilidades en aplicaciones web JAVA. En la primera parte se hizo una breve presentación sobre los avances y el adelanto actual de la revisión automatizada de código y su implementación en el desarrollo de proyectos. Luego se describió cada una de las técnicas utilizadas para realizar análisis estático y cuáles son los usos que se les pueden dar a estas herramientas, además se especifican las ventajas y desventajas que ofrecen al implementarse dentro del proceso de desarrollo de sistemas. Con esa base se procede a explicar las características y funcionalidades de SonarQube, por ser el

software propuesto por el ANCERT como una alternativa viable para ser implementada en sus desarrollos.[1]

- (Aguilera Díaz, 2014), en su investigación: "Analizando la calidad del código Java con Sonar" busca conocer una herramienta muy útil sobre todo para el rol de "Jefe de Desarrollo" ya que de un sólo vistazo da valiosa información si está bien escrito el código de un proyecto. Llegando a la siguiente conclusión esta herramienta nos facilita que nuestros proyectos mantengan una calidad óptima.[2]

- (Dávila Murillo, 2010), en su investigación: *"Software para controlar la calidad en el desarrollo de software SONAR"* busca utilizar Sonar y su funcionalidad para que las empresas cubran su necesidad de poder contar con Herramientas para el control de calidad en el código fuente de sus desarrollos de Software. Llegando a la siguiente conclusión en el mercado nacional no existen las suficientes herramientas para controlar la calidad de los programas desarrollados, sin embargo se cuenta con productos como Sonar, el cual llega a cubrir las necesidades importantes de la calidad del Software. Sonar es una herramienta muy valiosa sobre todo para el trabajo que desempeña el Jefe o Líder de Desarrollo, ya que de una sola revisión puede ver toda la información del código de un proyecto, puede identificar cuáles son las fallas que se han detectado, cual es porcentaje de cumplimiento de reglas y el nivel de complejidad tanto de las clases como de los métodos que lo componen. De esta

[1]http://openaccess.uoc.edu/webapps/o2/bitstream/10609/43263/3/gnupabloTFM0715memoria.pdf
[2]https://www.adictosaltrabajo.com/tutoriales/utilizando-sonar/

manera se puede controlar y evaluar el desarrollo que sus programadores implementen.[3]

- (Bavera, 2006), en su investigación: *"Compilación y certificación de código mediante análisis estático de flujo de control y de datos"* realizando el siguiente aporte: Se diseñó y desarrollo un prototipo de compilador certificante basado en análisis estático de flujo de control y de datos. Este compilador permite certificar propiedades de seguridad de gran importancia. Por ejemplo, certifica la ausencia de accesos inválidos a los arreglos. CCMini toma como entrada un subconjunto extendido de C y genera un ´árbol sintáctico abstracto anotado con información del estado del programa en cada punto. CCMini asegura que solo serán descartados aquellos programas que son ciertamente inseguros. Si no puede determinar la seguridad de un programa inserta verificaciones dinámicas que garantizan la ejecución segura del programa. Esto no solo ocurre por las limitaciones de un análisis estático particular sino porque los problemas a resolver son no computables, como por ejemplo, garantizar la seguridad al eliminar la totalidad de array-bound checking dinámicos.[4]

- (Alba Castro, 2008), en su investigación: "Certificación automática de propiedades de seguridad de código fuente Java en lógica de reescritura" realizando el siguiente aporte por lo que se sabe, el enfoque desarrollado es la primera aproximación correcta, y completa de una herramienta completamente automática para la verificación de

[3] http://repositorio.ug.edu.ec/bitstream/redug/6899/1/Tesis%20Completa-289-2011.pdf
[4] http://dc.exa.unrc.edu.ar/staff/fbavera/papers/tesis_maestria_FB.pdf

propiedades aritméticas basadas en tipos no verificables por un compilador estándar, de programas Java (código fuente). La completitud es relativa a la abstracción, pero puede obviarse al mejorar la aproximación. La escalabilidad a código de tamaño y complejidad industriales depende de las abstracciones disponibles, para el cual los dominios abstractos EvenOdd y Mod4 no tenían la precisión suficiente. Esto quiere decir que para certificar un programa Java dado, es posible que se necesite definir un nuevo dominio abstracto. Si se quiere certificar la política de resultados EvenOdd, el nuevo dominio posiblemente sea una refinación de los existentes EvenOdd, Mod4 y AbstLeqN.[5]

2.2. BASES TEÓRICAS

JAVA

El lenguaje Java fue creado por Sun Microsystems Inc., Aparece en el año 1995 y debe, en gran medida, su popularidad al éxito del servicio WWW. Se creó en su origen para que fuese un lenguaje multiplataforma. Para ello se compila en un código intermedio: bytecode y necesita de una máquina virtual que lo ejecute. Normalmente, no utiliza código nativo, es decir, no se puede ejecutar directamente por el procesador. (Bernal Bermúdez, 2012)

Java es un lenguaje de programación con el que podemos realizar cualquier tipo de programa. En la actualidad es un lenguaje muy extendido

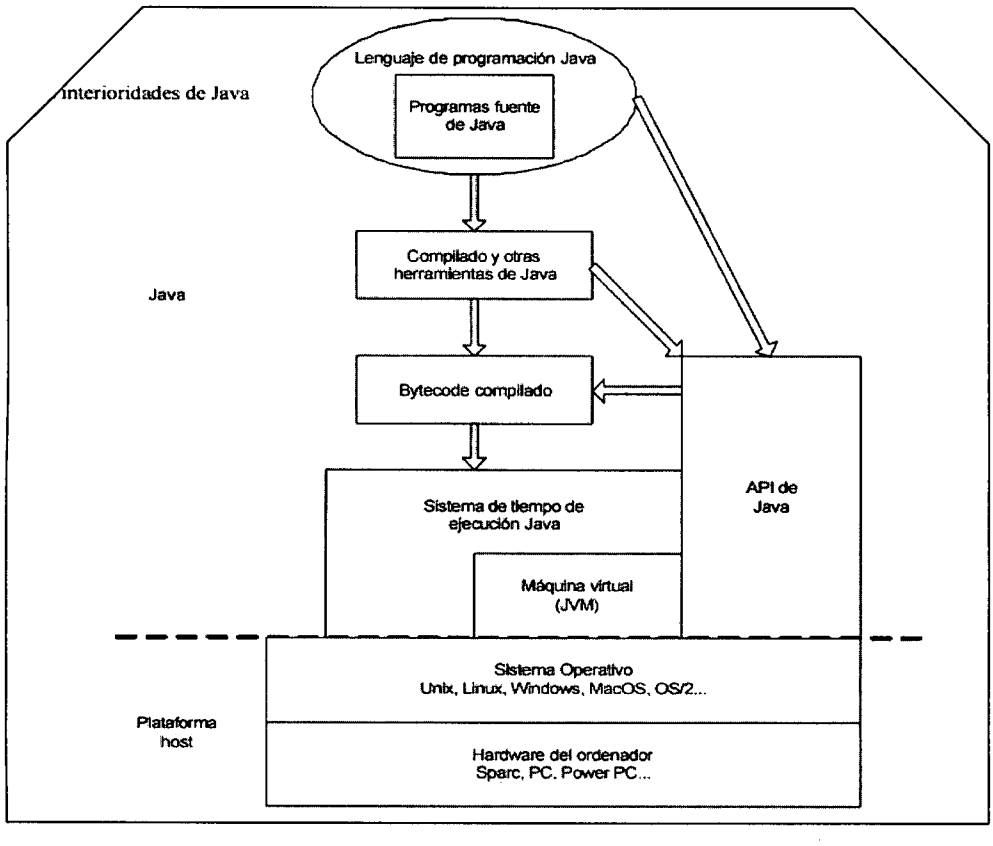
[5]https://riunet.upv.es/bitstream/handle/10251/12247/tesina%20Mauricio%20Alba-Castro.pdf?sequence=1&isAllowed=y

y cada vez cobra más importancia tanto en el ámbito de Internet como en la informática en general. Está desarrollado por la compañía Sun Microsystems con gran dedicación y siempre enfocado a cubrir las necesidades tecnológicas más punteras. (Alvarez, 2001)

El significado de Java, tal y como se le conoce en la actualidad, es el lenguaje de programación y un entorno de ejecución de programas escritos en Java. Al contrario de los compiladores tradicionales , que convierten el código fuente en instrucciones a nivel de máquina, el compilador Java traduce el código fuente Java en instrucciones que son interpretadas por la máquina virtual de Java (JVM, Java Virtual Machine). A diferencia de C y C++ en los que está inspirado. *Java es un lenguaje interpretado.* (Cruz Vílchez, 2012)

(Párraga García, 2003), no es sólo un lenguaje de programación, Java es además un sistema de tiempo de ejecución, un juego de herramientas de desarrollo y una interfaz de programación de aplicaciones (API). Todos estos elementos así como las relaciones establecidas entre ellos se esquematizan en la siguiente figura.

Figura 1. Las interioridades de Java



Fuente: (Párraga García, 2003)

CARACTERÍSTICAS DE JAVA COMO LENGUAJE DE PROGRAMACIÓN

De acuerdo a la página web (JavaSun, 2014), las características principales que nos ofrece Java son:

- Lenguaje simple: Se lo conoce como lenguaje simple porque viene de la misma estructura de C y C++; ya que C++ fue un referente para la

creación de java por eso utiliza determinadas características de C++ y se han eliminado otras.

- Orientado a objeto: Toda la programación en java en su mayoría está orientada a objeto, ya que al estar agrupados en estructuras en estructuras encapsuladas es más fácil su manipulación.

- Distribuido: Permite abrir sockets, establecer y aceptar conexiones con los servidores o clientes remotos; facilita la creación de aplicaciones distribuidas ya que proporciona una colección de clases para aplicaciones en red.

- Robusto: Es altamente fiable en comparación con c, se han eliminado muchas características con la aritmética de punteros, proporciona numerosas comprobaciones en compilación y en tiempo de ejecución.

- Seguro: La seguridad es una característica muy importante en java ya que se han implementado barreras de seguridad en el lenguaje y en el sistema de ejecución de tiempo real.

- Indiferente a la arquitectura: Java es compatible con los más variados entornos de red, cualesquiera sean estos desde Windows 95, Unix aWindows Nt y Mac, para poder trabajar con diferentes sistemas operativos.

  Java es muy versátil ya que utiliza byte-codes que es un formato intermedio que sirve para transportar el código eficientemente o de diferentes plataformas (Hardware - Software).

- Portable: Por ser indiferente a la arquitectura sobre la cual está trabajando, esto hace que su portabilidad sea muy eficiente, sus programas son iguales en cualquiera de las plataformas, ya que java especifica tamaños básicos, esto se conoce como la máquina virtual de Java.

- Interpretado y compilado a la vez: Java puede ser compilado e interpretado en tiempo real, ya que cuando se construye el código fuente este se transforma en una especie de código de máquina.

- Multihebra o Multihilos: Java tiene una facilidad de cumplir varias funciones al mismo tiempo, gracias a su función de multihilos ya que por cada hilo que el programa tenga se ejecutaran en tiempo real muchas funciones al mismo tiempo.

- Dinámico: El lenguaje java es muy dinámico en la fase de enlazado, sus clases solamente actuaran en medida en que sean requeridas o necesitadas con esto permitirá que los enlaces se puedan incluir incluso desde fuentes muy variadas o desde la red.

CODIGO FUENTE

El código fuente no es otra cosa que un texto escrito, simple. Un texto similar al que usted genera cuando usa el "block de notas" de su sistema Windows; un texto plano, es decir, sin formatos rigurosamente ceñido al vocabulario y a las reglas semánticas y de sintaxis propias del lenguaje elegido por el programador.

El código fuente es texto simple, capaz de ser leído por cualquier editor de textos y lo que es más importante, comprensible por cualquier programador que conozca el "idioma" utilizado. En él están escritas las instrucciones que deberá realizar la computadora, según la sintaxis de un lenguaje de programación. (Casañas, 2014)

Conjunto de reglas y normas propios de la aplicación que es utilizada para programar, pero siendo siempre entendible por un usuario que tenga estos conocimientos específicos. (Zamora Herrera, 2012)

Texto desarrollado en un lenguaje de programación y que debe ser compilado o interpretado para poder ejecutarse en un ordenador, también llamado computadora. Asimismo facilita la comunicación entre distintos dispositivos digitales. Las letras, los números, los símbolos y los acentos tienen su correspondiente número ASCII, de manera que programas desarrollados por diferentes personas y equipos fabricados por una u otra compañía pueden entender el texto de la misma forma. (Pérez Porto & Gardey, 2012)

Texto escrito en un lenguaje de programación específico y que puede ser leído por un programador. Debe traducirse a lenguaje máquina para que pueda ser ejecutado por la computadora o a bytecode para que pueda ser ejecutado por un intérprete. Este proceso se denomina compilación. Acceder al código fuente de un programa significa acceder a los algoritmos desarrollados por sus creadores. Es la única manera de modificar eficaz y eficientemente un programa.

Liberar un código fuente significa compartir ese texto con cualquier persona que lo desee, esto implica que cualquiera puede analizarlo, copiarlo o modificarlo. Las aplicaciones que liberan sus códigos suelen ser gratuitas bajo licencia GNU. La liberación de un código de un programa implica ciertos riesgos en cuando a la seguridad del mismo, pues su funcionamiento exacto queda expuesto. El código fuente no suele liberarse para las aplicaciones comerciales y no puede obtenerse el código fuente original a través de los programas ya compilados. (Alegsa, 2010)

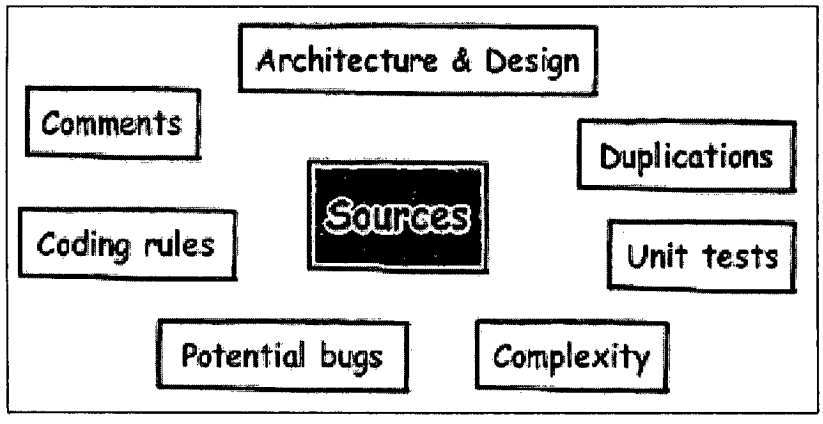
SONARQUBE

(Ospina Delgado, 2015), es una plataforma de código abierto usada por los equipos de desarrollo para controlar la calidad del código. Sonar fue desarrollado con el principal objetivo de hacer accesible la administración de la calidad del código con un mínimo esfuerzo. Como tal, Sonar contiene en su núcleo de funcionalidades un analizador de código, una herramienta de reportes, un módulo que detecta defectos y una función para regresar los cambios realizados en el código.

(Cambell, 2016), es una plataforma de gestión de calidad de código abierto, dedicado a analizar y medir continuamente la calidad técnica, de la cartera de proyectos de método.

(Garcia, 2013), es una herramienta para gestionar la calidad del código, la herramienta controla la calidad en 7 ejes:

Figura 2. Ejes de calidad



Fuente: (Garcia, 2013)

ARQUITECTURA DE SONARQUBE

(Gigleux, 2016), indica que la plataforma SonarQube es de 4 componentes:

1. Una SonarQube servidor empezando 3 procesos principales:

    a. Un servidor web para desarrolladores, administradores para navegar instantáneas de calidad y configuración de la instancia SonarQube

    b. Un servidor de búsqueda basado en Elasticsearch realizar una copia de búsquedas de la interfaz de usuario.

    c. Un servidor Compute Engine a cargo de los informes de análisis de código de procesamiento y almacenamiento en la base de datos SonarQube

2. Una SonarQube base de datos a almacenar:

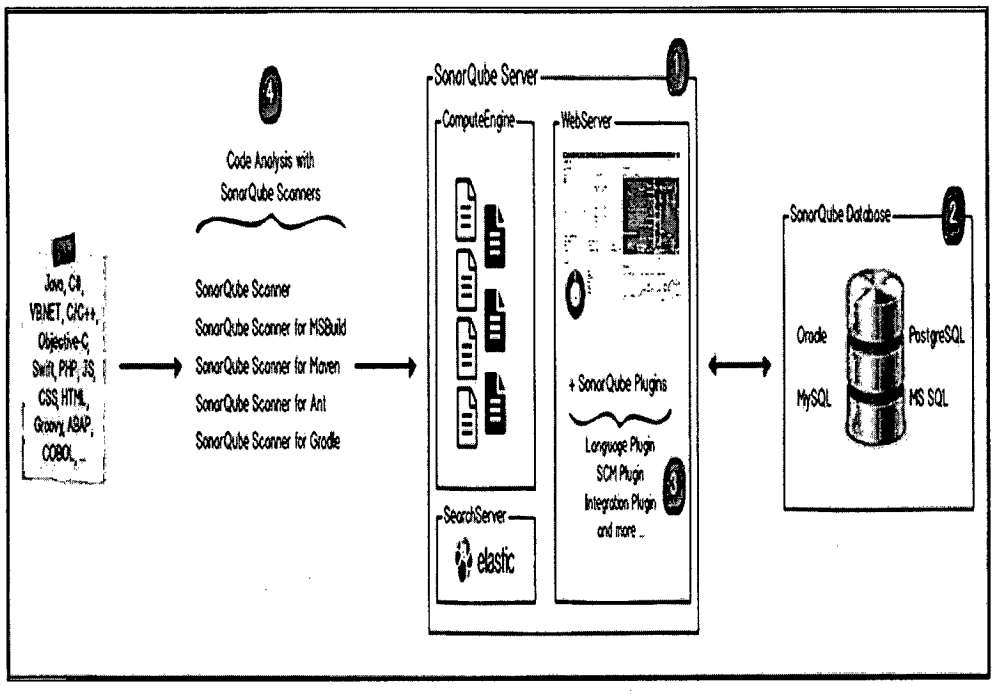
   - La configuración de la instancia SonarQube (seguridad, plugins configuración, etc.)

   - Las instantáneas de calidad de los proyectos, vistas, etc.

3. Múltiples SonarQube plugins instalado en el servidor, posiblemente incluyendo el lenguaje, SCM, plugins de integración, de autenticación y de gobernabilidad.

4. Uno o más SonarQube Escáneres que se ejecuta en su construcción Servidores / continuos de integración para analizar proyectos.

Figura 3. Arquitectura de SonarQube



Fuente: (Gigleux, 2016)

INTEGRACIÓN DE SONARQUBE

(Gigleux, 2016), indica que el siguiente esquema muestra cómo SonarQube se integra con otras herramientas de ALM y donde se utilizan los diversos componentes de SonarQube:

1. Los desarrolladores de código en sus entornos de desarrollo y utilizar SonarLint para ejecutar análisis local.

2. Los desarrolladores empujan su código en su SMC favorita: Git, SVN, TFVC, etc.

3. El servidor de integración continua desencadena una acumulación automática, y la ejecución del escáner SonarQube requerido para ejecutar el análisis SonarQube.

4. El informe de análisis se envía al servidor SonarQube para su procesamiento.

5. SonarQube procesos de servidor y almacena los resultados del informe de análisis en la base de datos SonarQube, y muestra los resultados en la interfaz de usuario.

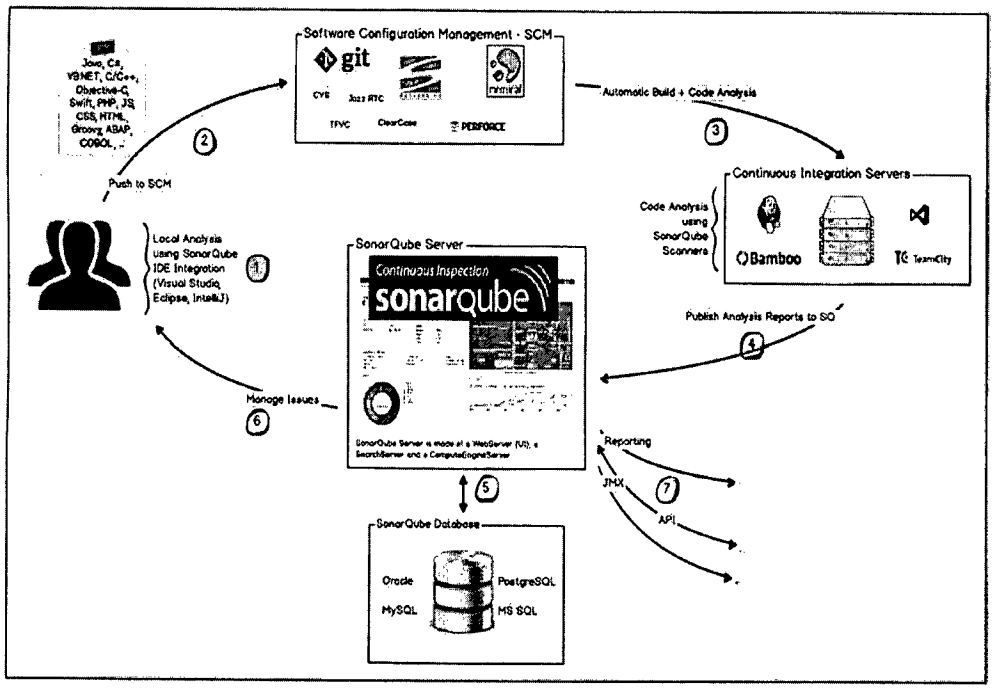
6. Los desarrolladores opinión, comentario, desafían a sus problemas para gestionar y reducir su deuda técnica a través de la interfaz de usuario SonarQube.

7. Los gerentes reciben informes de análisis.

   Ops utilizan las API para automatizar los datos de configuración y extraer de SonarQube.

OPS La utilización JMX para monitorear SonarQube servidor.

Figura 4. Integración de SonarQube



Fuente: (Gigleux, 2016)

REQUISITOS DE HARDWARE PARA SONARQUBE

- El servidor SonarQube requiere al menos 2 GB de RAM para ejecutar de manera eficiente y 1 GB de memoria RAM libre para el sistema operativo.

- La cantidad de espacio en disco que necesita dependerá de la cantidad de código que analizar con SonarQube. A modo de ejemplo, SonarQube.com la instancia pública de SonarQube, tiene más de 30 millones de líneas de código bajo análisis con 4 años de historia. SonarQube.com se está ejecutando actualmente en un m4.large

Amazon EC2 ejemplo, utilizando alrededor de 10 Gb de espacio en disco. Maneja más de 800 proyectos que tienen más o menos 3M cuestiones abiertas. SonarQube.com se está ejecutando en PostgreSQL 9.5 y está utilizando aproximadamente 15 GB de espacio en el disco.

- SonarQube debe estar instalado en los discos duros que tienen un excelente rendimiento de lectura y escritura. Lo más importante, la carpeta "data" alberga los índices Elasticsearch en la que se hará una enorme cantidad de E / S cuando el servidor está en funcionamiento. Gran lectura y escritura rendimiento del disco duro serán por lo tanto tienen un gran impacto en el rendimiento general del servidor SonarQube. (Gaudin, 2016)

MEDIDA DE LA CALIDAD DE TU CÓDIGO

(Caro, 2010), estas tres variables son básicamente las siguientes:

- Análisis de la calidad del código de las aplicaciones.
- Profiling de aplicaciones para detectar cuellos de botella y problemas de memoria (memory leaks).
- Ejecución de pruebas de carga/estrés que estudie la escalabilidad que presenta la infraestructura tanto hardware como software.

JENKINS

Es un motor de automatización con un ecosistema plug-in sin precedentes para apoyar todas sus herramientas favoritas en sus tuberías de suministro,

su objetivo es la integración continua, pruebas automatizadas, o la entrega continua. (Jenkins, 2014)

Es un software de integración continua open source escrito en Java. Está basado en el proyecto Hudson y es, dependiendo de la visión, un fork del proyecto o simplemente un cambio de nombre.

Jenkins proporciona integración continua para el desarrollo de software. Es un sistema corriendo en un servidor que es un contenedor de servlets, como Apache Tomcat. Soporta herramientas de control de versiones como CVS, Subversion, Git, Mercurial, Perforce y Clearcase y puede ejecutar proyectos basados en Apache Ant y Apache Maven, así como scripts de shell y programas batch de Windows. (Kawaguchi Kohsuke, 2011)

Jenkins es un servidor de integración continua, gratuito, open-source y actualmente uno de los más empleados para esta función.

Esta herramienta, proviene de otra similar llamada Hudson, ideada por Kohsuke Kawaguchi, que trabajaba en Sun. Unos años después de que Oracle comprara Sun, la comunidad de Hudson decidió renombrar el proyecto a Jenkins, migrar el código a Github y continuar el trabajo desde ahí. No obstante, Oracle ha seguido desde entonces manteniendo y trabajando en Hudson. (García Oterino, 2014)

2.3. DEFINICIÓN DE TÉRMINOS BÁSICOS

Considerando el (Diccionario informático, 2011), definimos los siguientes términos.

- Alfanumérico: Conjunto de letras, números y otros símbolos, como signos de puntuación o símbolos matemáticos. Hace referencia a los caracteres del teclado y al conjunto de caracteres disponibles para las diferentes operaciones de transferencia de datos del ordenador.
- Archivo: Se suele usar como sinónimo de fichero. En el mundo de los compresores, es frecuente hacer una diferenciación, diciendo que varios ficheros originales se comprimen y se empaquetan en un archivo comprimido.
- Binario: sistema de numeración en base 2, de modo que sólo hay dos dígitos posibles: el 0 y el 1. Para formar números "grandes", se usan varios dígitos binarios, que representan cada una de las potencias de 2. Por ejemplo, el número decimal 13 se representa 1101 (1x8 + 1x4 + 0x2 + 1x1).
- Byte: Es la unidad básica de información. En la práctica, se puede considerar que un byte es la cantidad de espacio necesaria para almacenar una letra. Tiene múltiplos como el Kilobyte, Megabyte, Gigabyte y Terabyte. Internamente, corresponde a 8 bits.
- CD: Normalmente se refiere a Compact Disc. Otras abreviaturas relacionadas son: CD-ROM, CD-R y CD-RW. Un Compact Disc para ordenador será capaz de almacenar cerca de 650 Mb de información.

- Código máquina: es el lenguaje en el que el ordenador es capaz de reconocer órdenes. Los programas que se creen en otros lenguajes deberán traducirse previamente a Código Máquina (compilarse) o el ordenador no podrá entenderlos.

- Compilador: Aplicación informática que se usa para crear programas en un cierto lenguaje de programación. Convierte los programas creados en un lenguaje de programación al lenguaje interno del ordenador (código máquina). En los compiladores, todo el programa original (fuente) se convierte a código máquina en bloque, y el programa resultante (programa ejecutable) se puede en otro ordenador usar sin necesidad de recurrir otra vez al compilador. En los intérpretes, el programa fuente se convierte a código máquina, línea por línea, justo en el momento en que se pone a funcionar; no se crea ningún ejecutable, y por eso es necesario distribuir el programa fuente pero también el intérprete que es capaz de entenderlo.

- Hardware: La parte "que se puede tocar" de un ordenador: caja (y todo su contenido), teclado, pantalla, etc.

- HTML: HyperText Markup Language: el lenguaje de descripción de páginas habitual en Internet.

- IBM: Importante fabricante de ordenadores y de software (International Business Machines).

- Java: Lenguaje de programación orientado a objetos, basado en C++, cada día más extendido, especialmente a través de Internet. Pretende

ser un lenguaje totalmente portable entre distintos ordenadores, gracias a que no se compila a código máquina,sino a un lenguaje intermedio que luego es interpretado por la "máquina virtual Java", que sí es específica de cada plataforma. Esto le da una velocidad ligeramente inferior a la de los programas realizados en otros lenguajes compilados, como C++, a cambio de una mayor portabilidad (aparte de las mejoras que el lenguaje en sí incorpora sobre otros como C++).

- Procesador: el "cerebro" del ordenador. Su velocidad de trabajo se mide en Megahertzios (MHz) y su capacidad de proceso por el número de bits que es capaz de manejar a la vez (por ejemplo: 32 bits, o 64 bits).

- Programa: un conjunto de órdenes para un ordenador. Cuando se trata de un programa ya terminado que se compra, se suele hablar de una Aplicación Informática. Los programas se deben escribir en un cierto lenguaje de programación. Los lenguajes de programación que se acercan más al lenguaje humano que al del ordenador reciben el nombre de "lenguajes de alto nivel" (como Pascal); los que se acercan más al ordenador son los de "bajo nivel" (como el ensamblador). Lo más habitual es crear los programas en un lenguaje de alto nivel (llamado "fuente") y después convertirlos al lenguaje propio del ordenador ("compilarlos" para obtener un "ejecutable").

- Sistema operativo: Es una capa intermedia entre el ordenador y el usuario. Se podría considerar como un programa (normalmente de

gran tamaño) que toma el control del ordenador y que nos proporciona las utilidades básicas. Para usos más avanzados, necesitaremos instalar aplicaciones informáticas como bases de datos, hojas de cálculo, programas a medida, etc.

- Software: La parte "que no se puede tocar" de un ordenador: los programas y los datos.

CAPITULO III

MARCO METODOLÓGICO

3.1. METODOLOGÍA DE LA INVESTIGACIÓN

3.1.1. TIPO DE INVESTIGACIÓN

El presente estudio reúne las condiciones de tipo **aplicada**, según lo consultado en el libro de (Hernández Sampieri, Fernández Collado, & Baptista Lucio, 2014), porque aplicamos el Sonarqube 4.0 en la optimización del código Java.

3.1.2. NIVEL DE INVESTIGACIÓN

Asimismo de acuerdo a la naturaleza de nuestra investigación, teniendo la apreciación de (Hernández Sampieri, Fernández Collado, & Baptista Lucio, 2014) es **descriptivo**, porque buscamos características del código Java en su respectiva evaluación. Es decir, pretendemos medir y/o recoger información sobre las variables a las que se refieren.

3.1.3. MÉTODO DE INVESTIGACIÓN

El **deductivo** se utiliza en la revisión de nuestro marco teórico y en la presentación del informe final.

El **inductivo** se utiliza cuando se procesan y se analizan los datos obtenidos del cuestionario aplicado y en el análisis e interpretación

de los datos. (Hernández Sampieri, Fernández Collado, & Baptista Lucio, 2014)

3.1.4. DISEÑO DE INVESTIGACIÓN

Se aplica el método de diseño en sucesión o en línea, conocido también como método Pre Test – Post Test el cual consiste en:

- Medición de la variable dependiente antes de aplicar la variable independiente (Pre - Test).
- Aplicación de la variable independiente.
- Medición de la variable dependiente después de aplicar la variable independiente (Post- Test).

Se puede representar mediante la siguiente simbología: GE, O1, X, O2.

$$GE: \ O_1 \ \rightarrow \ x \ \rightarrow \ O_2$$

Dónde:

- **GE:** La muestra (Grupo experimental)
- **O_1:** Análisis de los resultados, antes de la aplicación del Sonarqube 4.0.
- **X:** Sonarqube 4.0.
- **O_2:** Análisis de los resultados, después del Sonarqube 4.0.

Al final de la investigación se establece diferencias entre O_1 y O_2 para determinar si existe mejoramiento o no en los indicadores expresados en la variable independiente.

3.2. POBLACIÓN Y MUESTRA

3.2.1. POBLACIÓN.

Está conformada por el equipo que elaboran proyectos de software de la consultora Everis, en total son 16.

Tabla 2. Población del equipo de software.

Descripción	Cantidad
Encargado del proyecto	1
Analistas	2
Analistas en base de datos	2
Programadores	8
Practicantes	3
Total	**16**

3.2.2. MUESTRA

Se estima siguiendo los criterios que ofrece la estadística, teniendo en cuenta el muestreo no probabilístico, por conveniencia. El tamaño de la muestra es 16 colaboradores que deben ser encuestados.

3.3. PROCESAMIENTO DE RECOLECCIÓN DE DATOS.

Encuesta.- Documento formado por un conjunto de preguntas que están redactadas de forma coherente, a través de la organización y secuencia. (Hernández Sampieri, Fernández Collado, & Baptista Lucio, 2014)

Análisis documental.- Es la relación de documentos sea libros, informes o páginas webs que los investigadores deben leer para realizar su marco teórico. (Hernández Sampieri, Fernández Collado, & Baptista Lucio, 2014)

Tabla 3. Instrumentos de recolección de datos.

Técnicas	Instrumentos	Agente
Encuesta	Cuestionarios	Para el equipo que elaboran proyectos de software de la consultora Everis.
Análisis documental	Fichas textuales	Para el marco teórico de nuestra investigación.

3.4. TRATAMIENTO DE LOS DATOS

3.4.1. PROCESAMIENTO Y PRESENTACIÓN DE DATOS

IBM SPSS Statistics Base.- (IBM, 2015), software de análisis estadístico que presenta las funciones principales necesarias para realizar el proceso analítico de principio a fin. Es fácil de utilizar e

incluye un amplio rango de procedimientos y técnicas para ayudarle a aumentar los ingresos, superar a la competencia, dirigir investigaciones y tomar mejores decisiones.

Proporciona las herramientas básicas de análisis estadístico para cada paso del proceso analítico.

- Una amplia gama de procedimientos estadísticos para dirigir análisis precisos.
- Técnicas incorporadas para preparar los datos para el análisis de forma rápida y sencilla.
- Funcionalidad completa de elaboración de informes para la creación de gráficas muy eficientes.
- Potentes funciones de visualización que muestran claramente el significado de sus conclusiones.
- Soporte a todos los tipos de datos, incluidos grandes conjuntos de datos.

3.4.2. MECANISMOS PARA EL PROCESAMIENTO Y PRESENTACIÓN DE DATOS

Los datos son presentados en tablas y figuras, interpretados mediante el análisis estadístico descriptivo que ofrece el SPSS.

3.4.3. MECANISMOS PARA EL ANÁLISIS E INTERPRETACIÓN DE RESULTADOS

Los datos son interpretados en:

- Tablas.
- Figuras elaborados en diagramas de barras y curvas de frecuencias.

Se realiza el procesamiento y posteriormente la presentación de los datos recolectados, del respectivo análisis e interpretación de los resultados.

CAPITULO IV

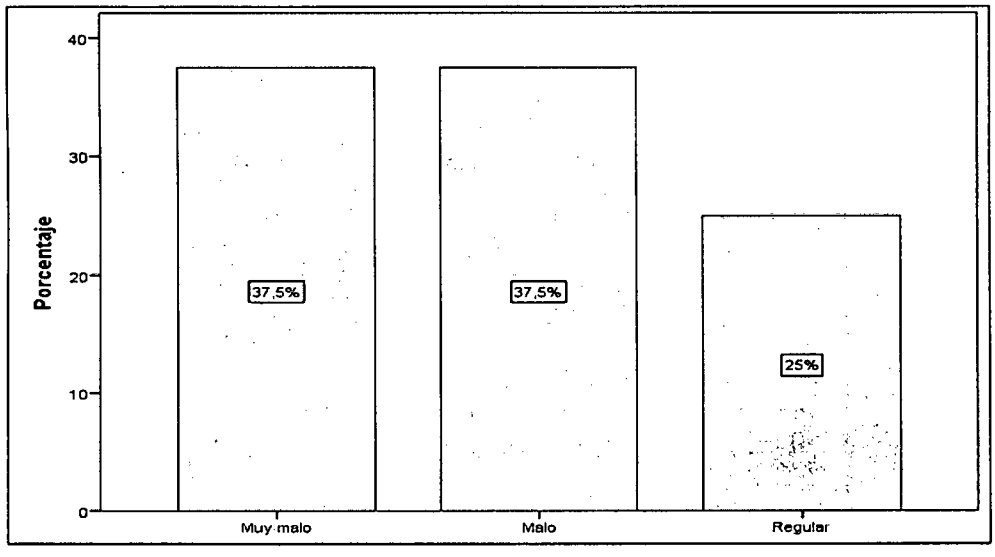
RESULTADOS

4.1. INTERPRETACIÓN DE RESULTADOS PRE - TEST

Tabla 4. Pre – Test ¿Cómo califica la funcionalidad del código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Muy malo	6	37,5	37,5
	Malo	6	37,5	75,0
	Regular	4	25,0	100,0
	Total	16	100,0	

Figura 5. Pre – Test ¿Cómo califica la funcionalidad del código Java?

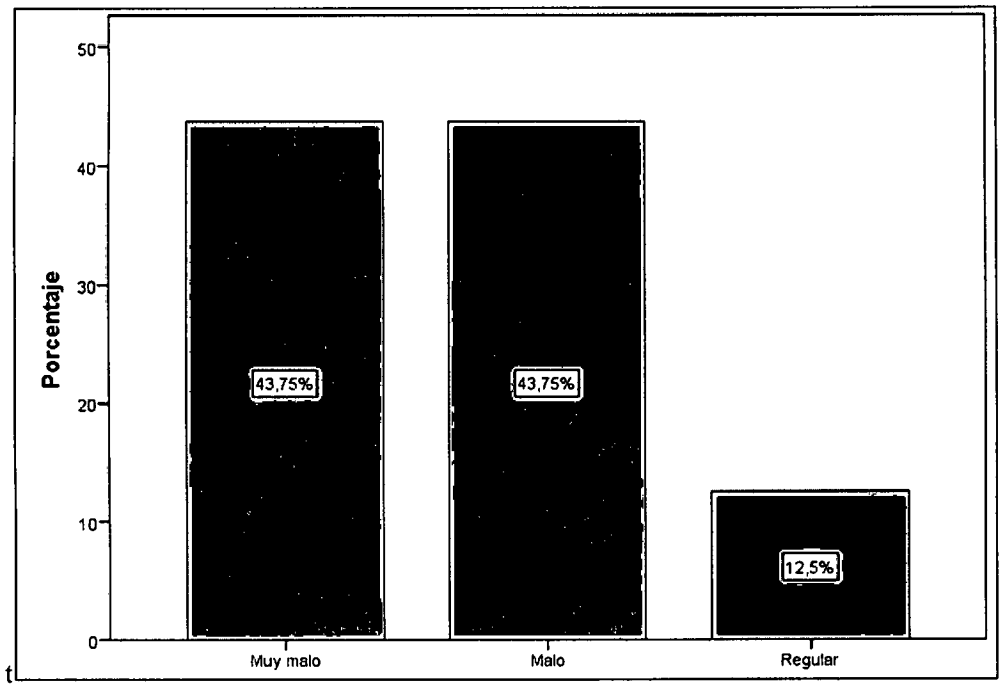


Lectura: El 37.5% califica la funcionalidad del código Java como Muy malo, el 37.5% Malo y el 25% Regular.

Tabla 5. Pre – Test ¿Cómo califica la confiabilidad del código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Muy malo	7	43,8	43,8
	Malo	7	43,8	87,5
	Regular	2	12,5	100,0
	Total	16	100,0	

Figura 6. Pre – Test ¿Cómo califica la confiabilidad del código Java?



Lectura: El 43.75% califica la confiabilidad del código Java como Muy malo, el 43.75% Malo y el 12.5% Regular.

Tabla 6. Pre – Test ¿Cómo califica la usabilidad del código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Muy malo	7	43,8	43,8
	Malo	8	50,0	93,8
	Regular	1	6,3	100,0
	Total	16	100,0	

Figura 7. Pre – Test ¿Cómo califica la usabilidad del código Java?

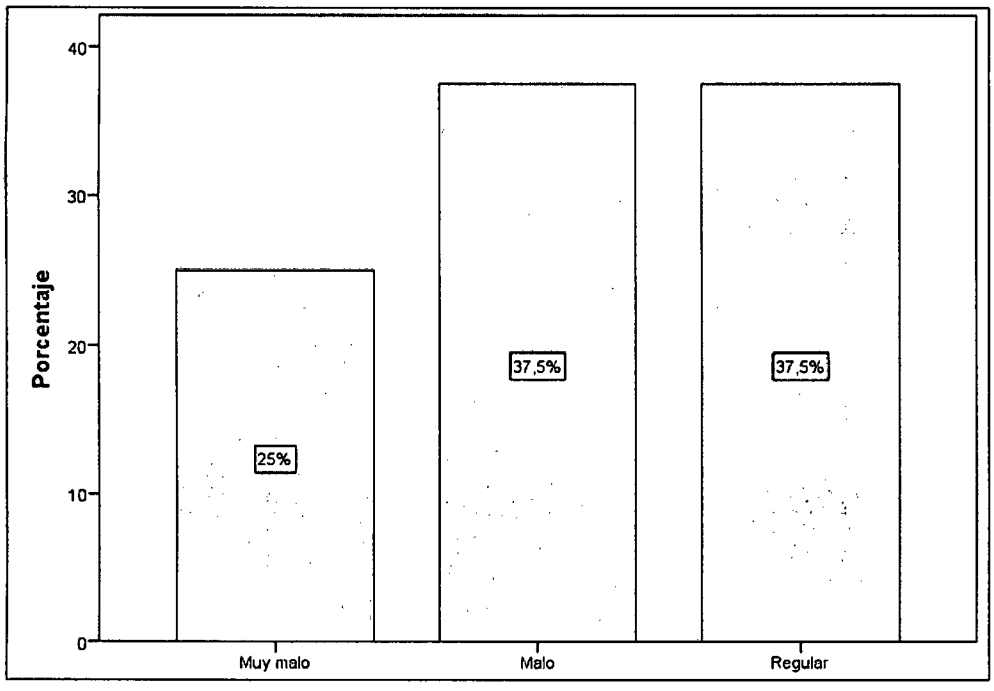


Lectura: El 43.75% califica la usabilidad del código Java como Muy malo, el 50% Malo y el 6.25% Regular.

Tabla 7. Pre – Test ¿Cómo califica la eficiencia del código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Muy malo	4	25,0	25,0
	Malo	6	37,5	62,5
	Regular	6	37,5	100,0
	Total	16	100,0	

Figura 8. Pre – Test ¿Cómo califica la eficiencia del código Java?

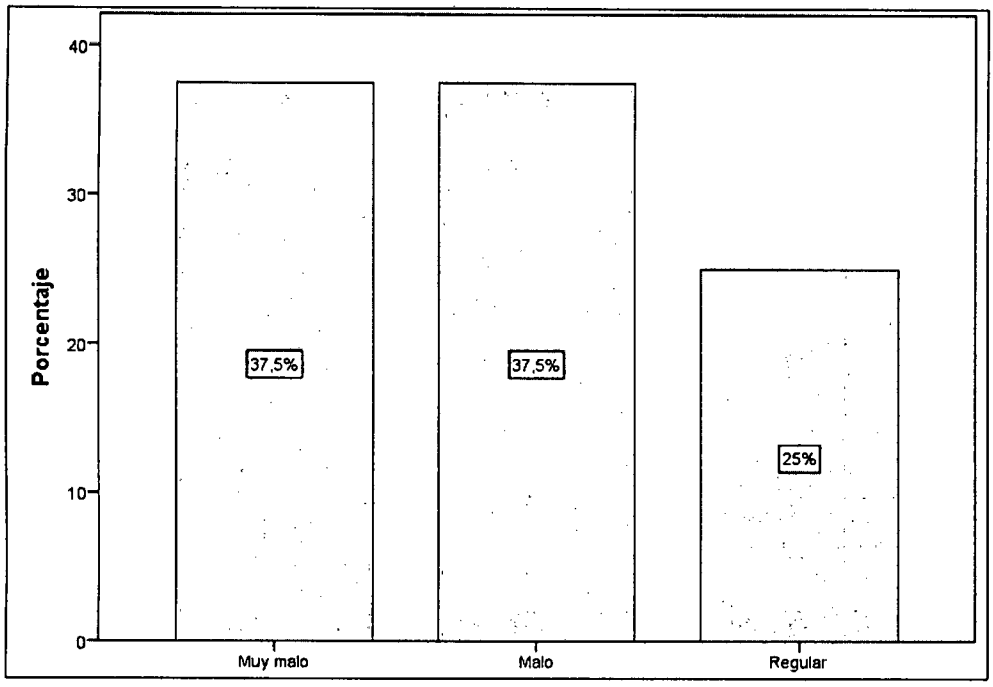


Lectura: El 37.5% califica la eficiencia del código Java como Malo, el 37.5% Regular y el 25% Muy malo.

Tabla 8. Pre – Test ¿Cómo califica la eficacia del código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Muy malo	6	37,5	37,5
	Malo	6	37,5	75,0
	Regular	4	25,0	100,0
	Total	16	100,0	

Figura 9. Pre – Test ¿Cómo califica la eficacia del código Java?

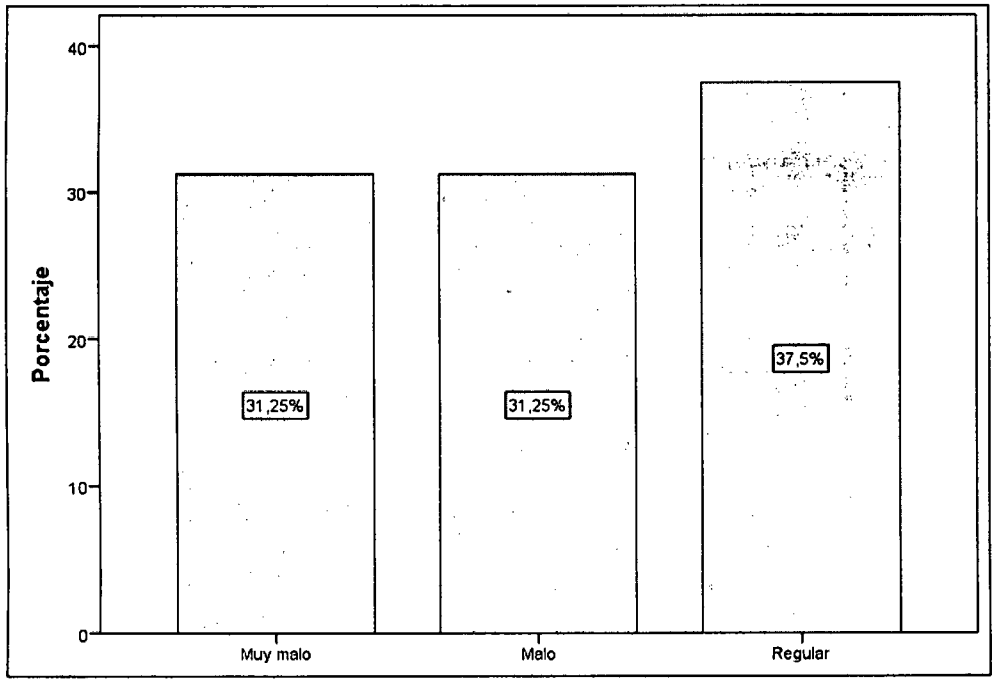


Lectura: El 37.5% califica la eficacia del código Java como Muy malo, el 37.5% Malo y el 25% Regular.

Tabla 9. Pre – Test ¿Cómo califica la portabilidad del código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Muy malo	5	31,3	31,3
	Malo	5	31,3	62,5
	Regular	6	37,5	100,0
	Total	16	100,0	

Figura 10. Pre – Test ¿Cómo califica la portabilidad del código Java?

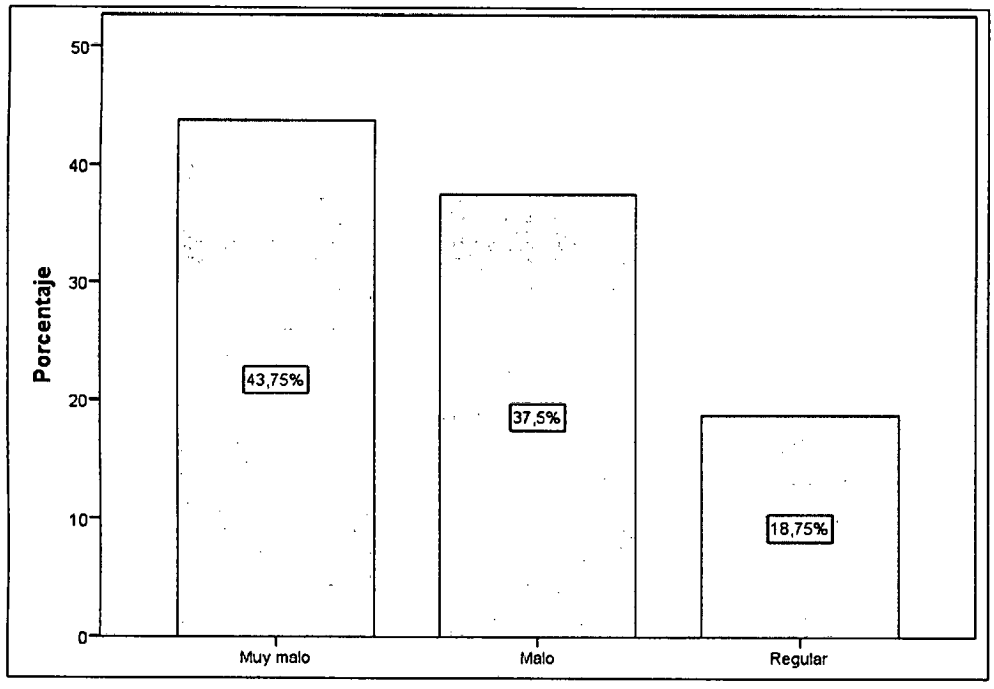


Lectura: El 37.5% califica la portabilidad del código Java como Regular, el 31.25% Muy malo y el 31.25% Malo.

Tabla 10. Pre – Test ¿Cómo califica los reportes emitidos en la certificación de código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Muy malo	7	43,8	43,8
	Malo	6	37,5	81,3
	Regular	3	18,8	100,0
	Total	16	100,0	

Figura 11. Pre – Test ¿Cómo califica los reportes emitidos en la certificación de código Java?

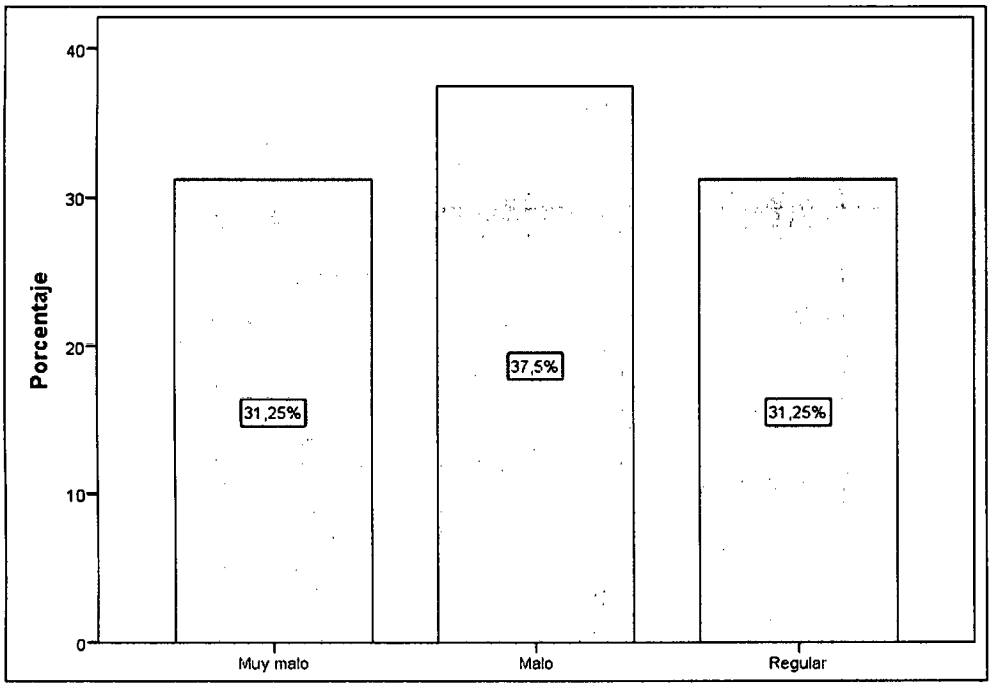


Lectura: El 43.75% califica los reportes emitidos en la certificación de código Java como Muy malo, el 37.5% Malo y el 18.75% Regular.

Tabla 11. Pre – Test ¿Cómo califica la verificación en la certificación de código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Muy malo	5	31,3	31,3
	Malo	6	37,5	68,8
	Regular	5	31,3	100,0
	Total	16	100,0	

Figura 12. Pre – Test ¿Cómo califica la verificación en la certificación de código Java?

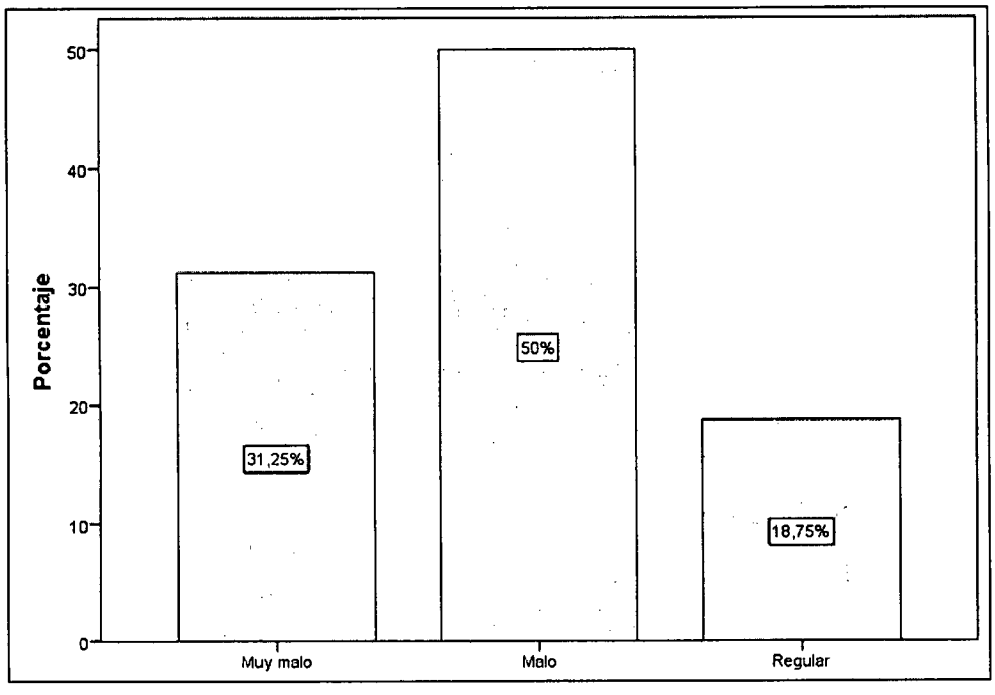


Lectura: El 37.5% califica la verificación en la certificación de código Java como Malo, el 31.25% Muy malo y el 31.25% Regular.

Tabla 12. Pre – Test ¿Cómo califica los procesos evaluación en la certificación de código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Muy malo	5	31,3	31,3
	Malo	8	50,0	81,3
	Regular	3	18,8	100,0
	Total	16	100,0	

Figura 13. Pre – Test ¿Cómo califica los procesos evaluación en la certificación de código Java?

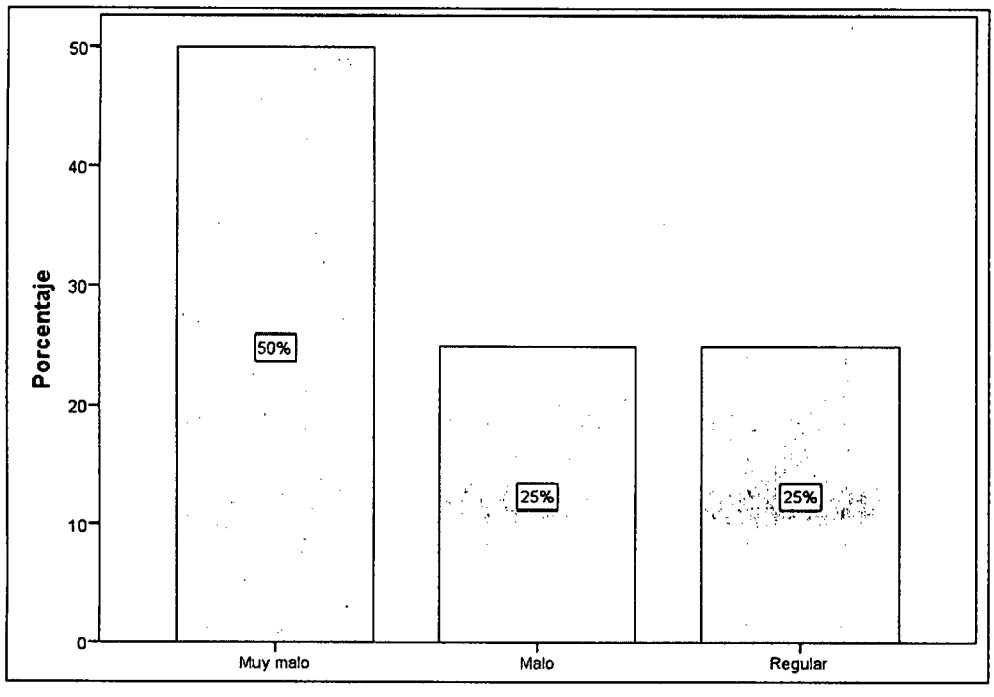


Lectura: El 50% califica los procesos evaluación en la certificación de código Java como Malo, el 31.25% Muy malo y el 18.75% Regular.

Tabla 13. Pre – Test ¿Cómo califica los recursos en la certificación de código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Muy malo	8	50,0	50,0
	Malo	4	25,0	75,0
	Regular	4	25,0	100,0
	Total	16	100,0	

Figura 14. Pre – Test ¿Cómo califica los recursos en la certificación de código Java?

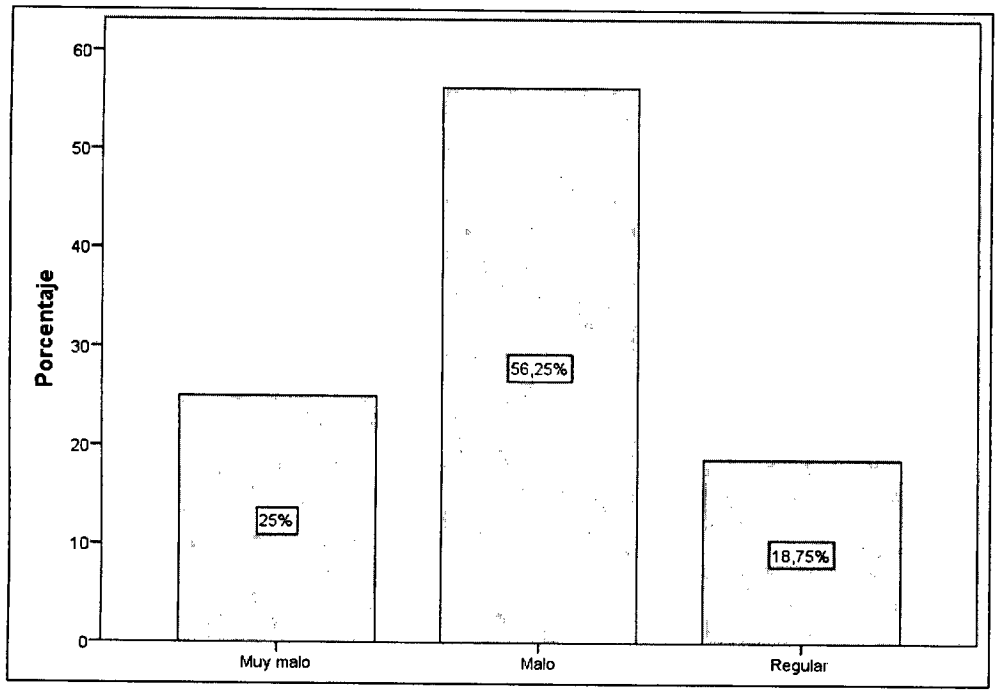


Lectura: El 50% califica los recursos en la certificación de código Java como Muy malo, el 25% Malo y el 25% Regular.

Tabla 14. Pre – Test ¿Cómo califica la satisfacción en la certificación de código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Muy malo	4	25,0	25,0
	Malo	9	56,3	81,3
	Regular	3	18,8	100,0
	Total	16	100,0	

Figura 15. Pre – Test ¿Cómo califica la satisfacción en la certificación de código Java?



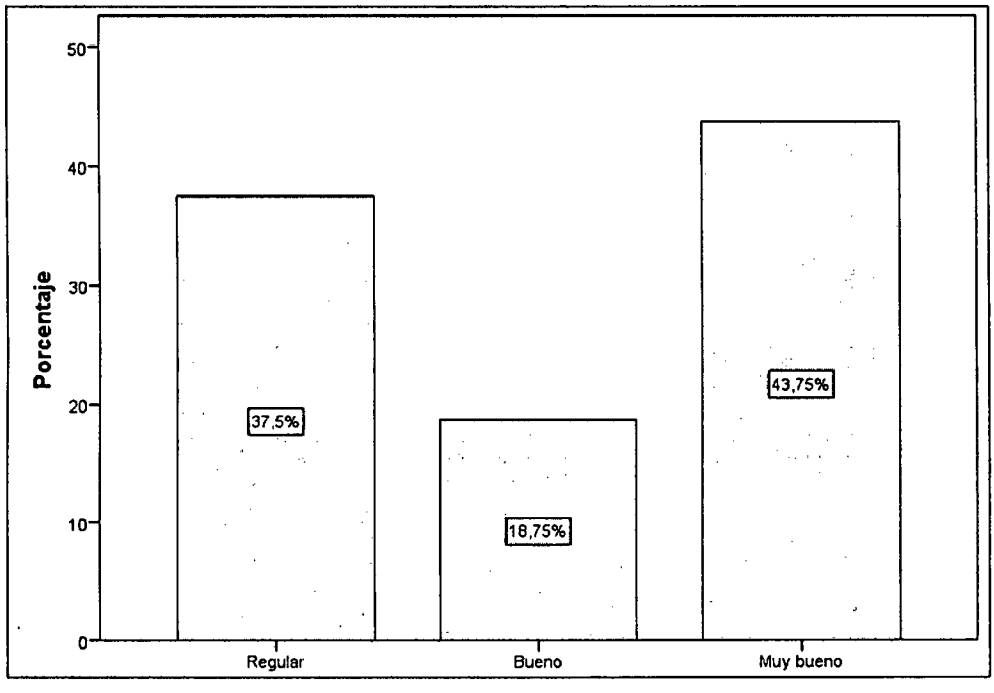
Lectura: El 56.25% califica la satisfacción en la certificación de código Java como Malo, el 25% Muy malo y el 18.75% Regular.

4.2. INTERPRETACIÓN DE RESULTADOS POST – TEST

Tabla 15. ¿Cómo califica la funcionalidad del SonarQube 4.0 en la certificación de código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Regular	6	37,5	37,5
	Bueno	3	18,8	56,3
	Muy bueno	7	43,8	100,0
	Total	16	100,0	

Figura 16. ¿Cómo califica la funcionalidad del SonarQube 4.0 en la certificación de código Java?

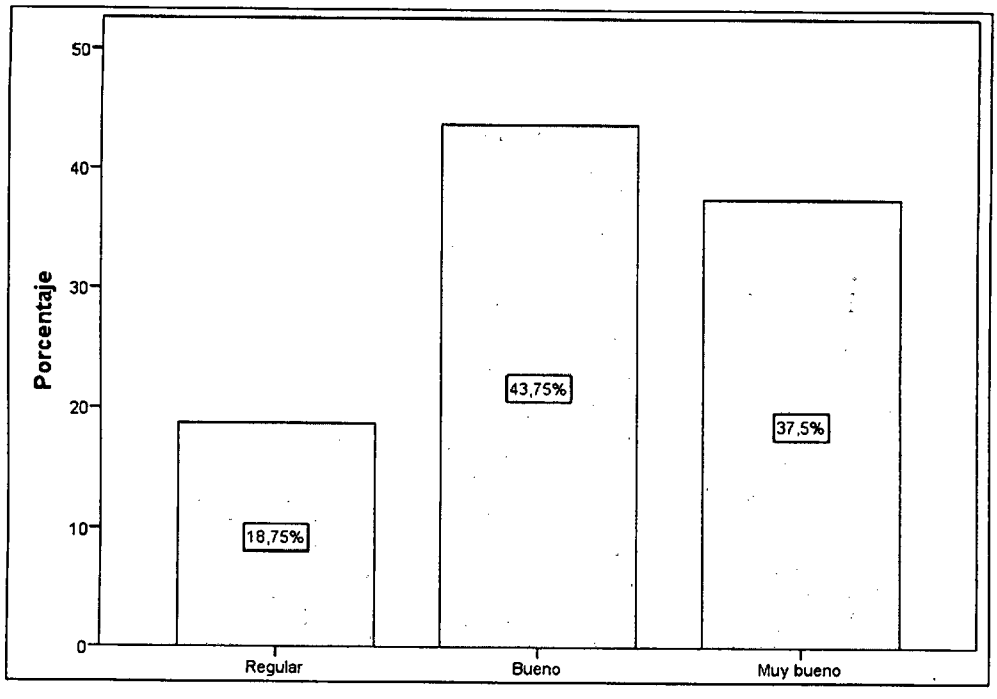


Lectura: El 43.75% califica la funcionalidad del SonarQube 4.0 en la certificación de código Java como Muy bueno, el 37.5% Regular y el 18.75% Bueno.

Tabla 16. ¿Cómo califica la confiabilidad del SonarQube 4.0 en la certificación de código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Regular	3	18,8	18,8
	Bueno	7	43,8	62,5
	Muy bueno	6	37,5	100,0
	Total	16	100,0	

Figura 17. ¿Cómo califica la confiabilidad del SonarQube 4.0 en la certificación de código Java?

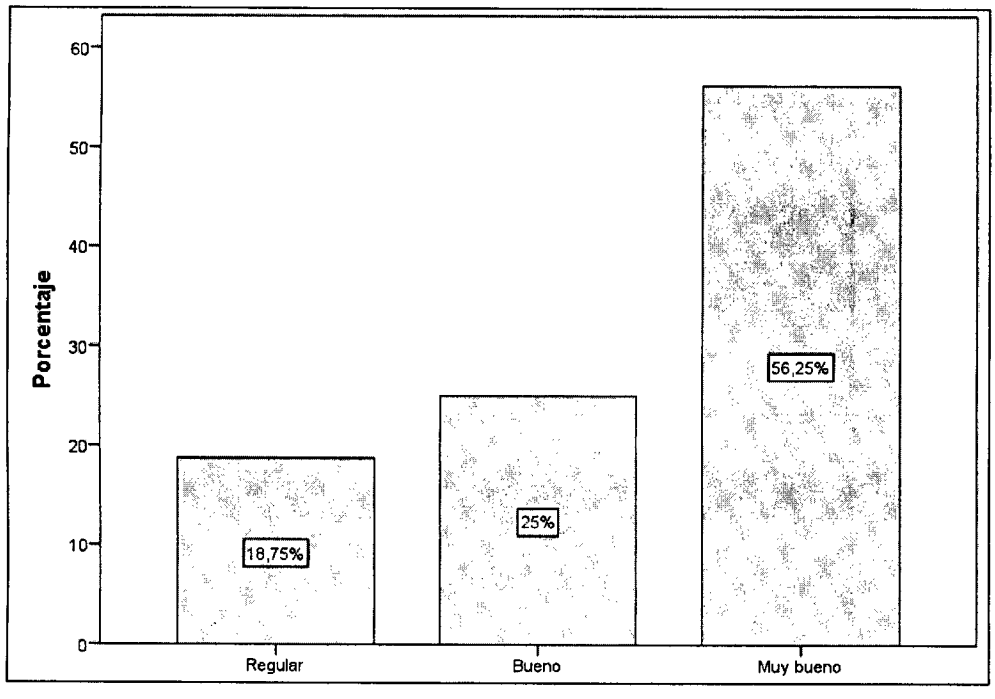


Lectura: El 43.75% califica la confiabilidad del SonarQube 4.0 en la certificación de código Java como Bueno, el 37.5% Muy bueno y el 18.75% Regular.

Tabla 17. ¿Cómo califica la usabilidad del SonarQube 4.0 en la certificación de código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Regular	3	18,8	18,8
	Bueno	4	25,0	43,8
	Muy bueno	9	56,3	100,0
	Total	16	100,0	

Figura 18. ¿Cómo califica la usabilidad del SonarQube 4.0 en la certificación de código Java?

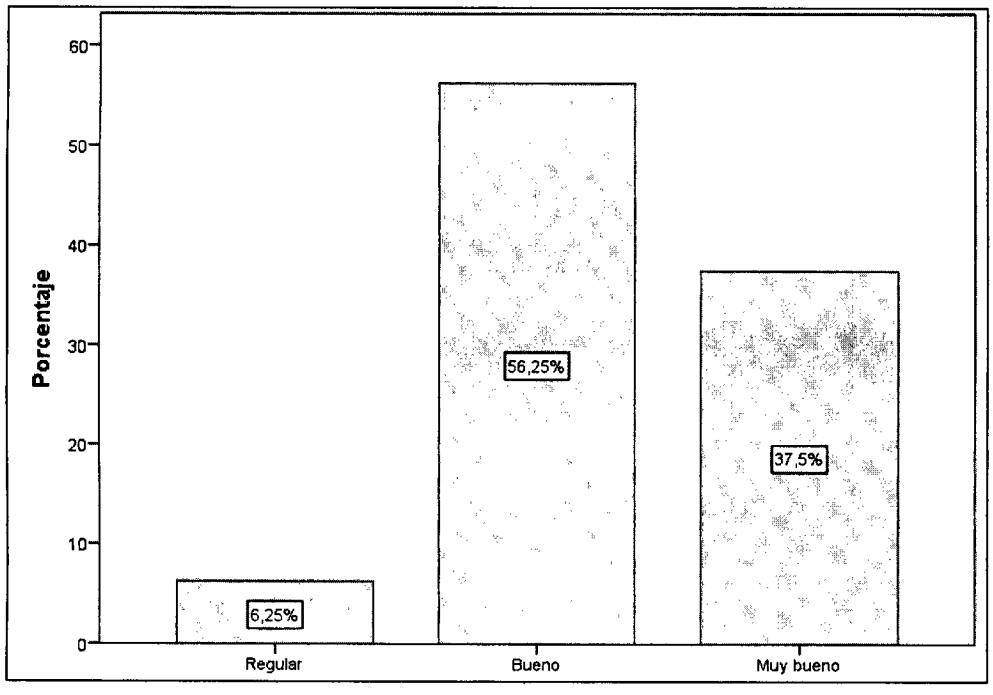


Lectura: El 56.25% califica la usabilidad del SonarQube 4.0 en la certificación de código Java como Muy bueno, el 25% Bueno y el 18.75% Regular.

Tabla 18. ¿Cómo califica la eficiencia del SonarQube 4.0 en la certificación de código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Regular	1	6,3	6,3
	Bueno	9	56,3	62,5
	Muy bueno	6	37,5	100,0
	Total	16	100,0	

Figura 19. ¿Cómo califica la eficiencia del SonarQube 4.0 en la certificación de código Java?

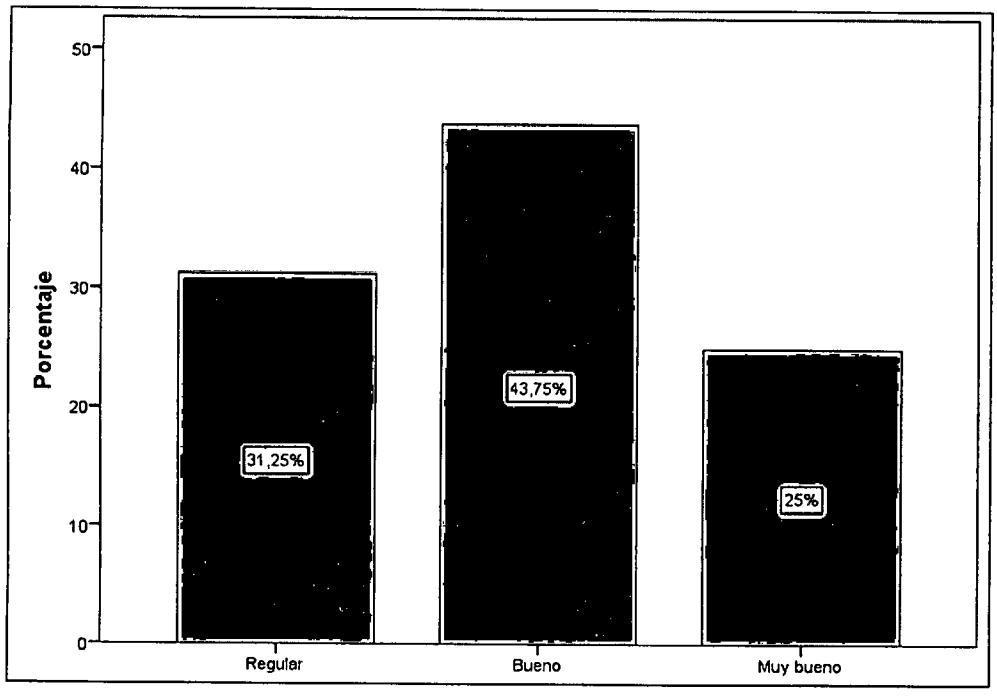


Lectura: El 56.25% califica la eficiencia del SonarQube 4.0 en la certificación de código Java como Bueno, el 37.5% Muy bueno y el 6.25% Regular.

Tabla 19. ¿Cómo califica la eficacia del SonarQube 4.0 en la certificación de código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Regular	5	31,3	31,3
	Bueno	7	43,8	75,0
	Muy bueno	4	25,0	100,0
	Total	16	100,0	

Figura 20. ¿Cómo califica la eficacia del SonarQube 4.0 en la certificación de código Java?

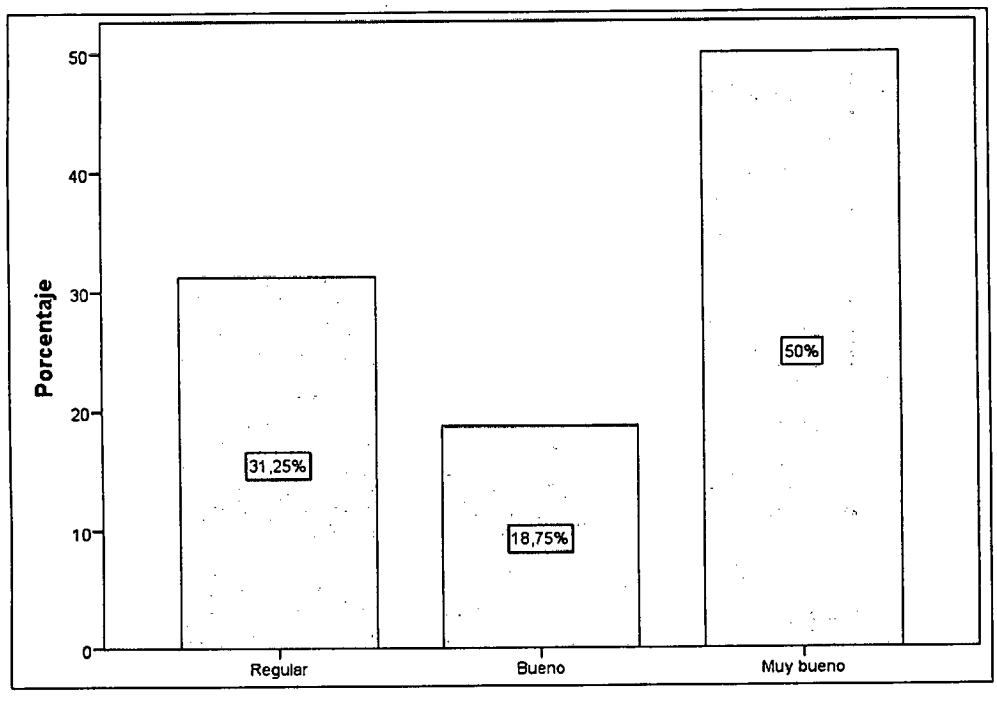


Lectura: El 43.75% califica la eficacia del SonarQube 4.0 en la certificación de código Java como Bueno, el 31.25% Regular y el 25% Muy bueno.

Tabla 20. ¿Cómo califica la portabilidad del SonarQube 4.0 en la certificación de código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Regular	5	31,3	31,3
	Bueno	3	18,8	50,0
	Muy bueno	8	50,0	100,0
	Total	16	100,0	

Figura 21. ¿Cómo califica la portabilidad del SonarQube 4.0 en la certificación de código Java?



Lectura: El 50% califica la portabilidad del SonarQube 4.0 en la certificación de código Java como Muy bueno, el 31.25% Regular y el 18.75% Bueno.

Tabla 21. ¿Cómo califica los reportes emitidos del SonarQube 4.0 en la certificación de código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Regular	1	6,3	6,3
	Bueno	6	37,5	43,8
	Muy bueno	9	56,3	100,0
	Total	16	100,0	

Figura 22. ¿Cómo califica los reportes emitidos del SonarQube 4.0 en la certificación de código Java?

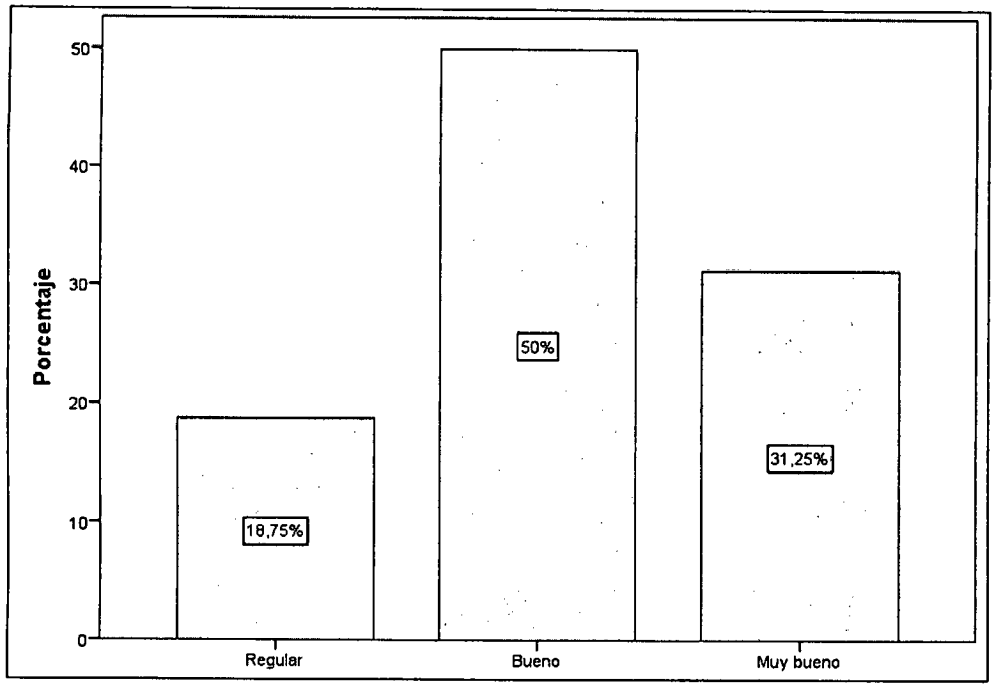


Lectura: El 56.25% califica los reportes emitidos del SonarQube 4.0 en la certificación de código Java como Muy bueno, el 37.5% Bueno y el 6.25% Regular.

Tabla 22. ¿Cómo califica la verificación del SonarQube 4.0 en la certificación de código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Regular	3	18,8	18,8
	Bueno	8	50,0	68,8
	Muy bueno	5	31,3	100,0
	Total	16	100,0	

Figura 23. ¿Cómo califica la verificación del SonarQube 4.0 en la certificación de código Java?



Lectura: El 50% califica la verificación del SonarQube 4.0 en la certificación de código Java como Bueno, el 31.25% Muy bueno y el 18.75% Regular.

Tabla 23. ¿Cómo califica los procesos del SonarQube 4.0 en la certificación de código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Regular	2	12,5	12,5
	Bueno	7	43,8	56,3
	Muy bueno	7	43,8	100,0
	Total	16	100,0	

Figura 24. ¿Cómo califica los procesos del SonarQube 4.0 en la certificación de código Java?

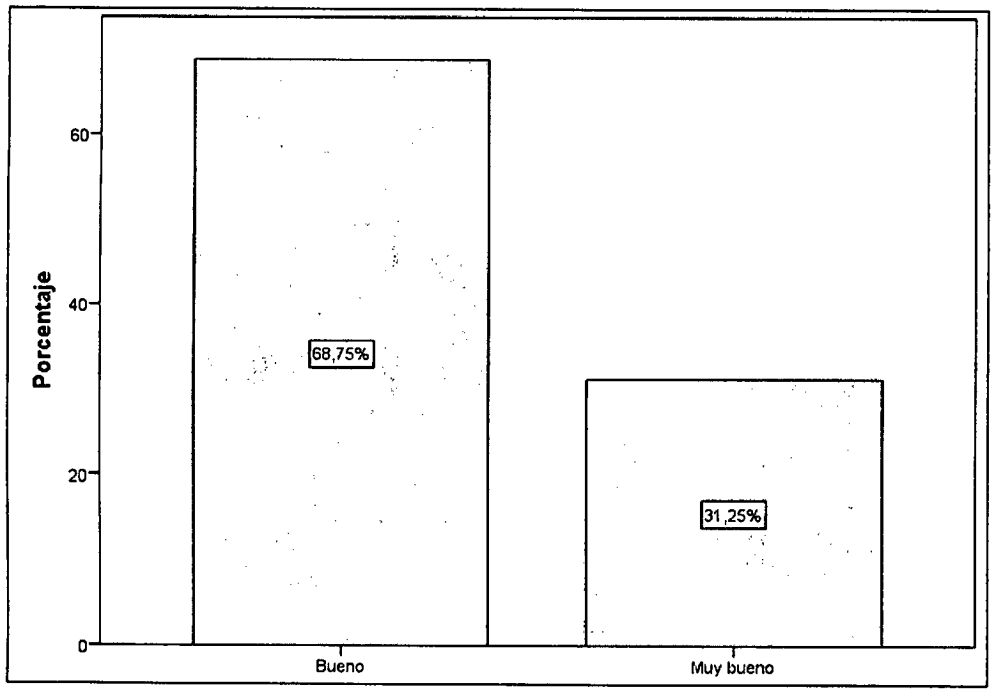


Lectura: El 43.75% califica los procesos del SonarQube 4.0 en la certificación de código Java como Bueno, el 43.75% Muy bueno y el 12.5% Regular.

Tabla 24. ¿Cómo califica los recursos del SonarQube 4.0 en la certificación de código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Bueno	11	68,8	68,8
	Muy bueno	5	31,3	100,0
	Total	16	100,0	

Figura 25. ¿Cómo califica los recursos del SonarQube 4.0 en la certificación de código Java?

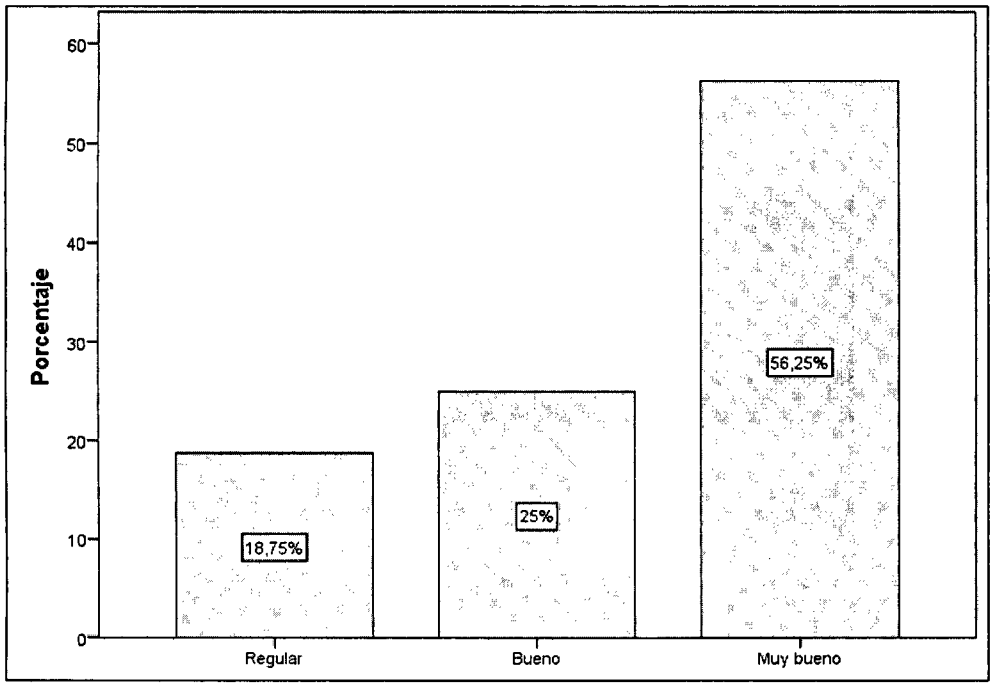


Lectura: El 68.75% califica los recursos del SonarQube 4.0 en la certificación de código Java como Bueno y el 31.25% Muy bueno.

Tabla 25. ¿Cómo califica la satisfacción del SonarQube 4.0 en la certificación de código Java?

		Frecuencia	Porcentaje	Porcentaje acumulado
Válidos	Regular	3	18,8	18,8
	Bueno	4	25,0	43,8
	Muy bueno	9	56,3	100,0
	Total	16	100,0	

Figura 26. ¿Cómo califica la satisfacción del SonarQube 4.0 en la certificación de código Java?



Lectura: El 56.25% califica la satisfacción del SonarQube 4.0 en la certificación de código Java como Muy bueno, el 25% Bueno y el 18.75% Regular.

4.3. PRUEBA DE HIPÓTESIS

4.3.1. FORMULACIÓN DE CUADRO DE VALORES DE INDICADORES

La contrastación de la hipótesis se realiza de acuerdo al diseño de investigación, el cual es conocido también como Pre – Test y Post – Test, que se representa mediante la siguiente simbología:

$$GE: \ O_1 \ \rightarrow \ x \ \rightarrow \ O_2$$

Donde:

GE: (Grupo experimental).

O_1: Observación 1, antes de la propuesta SonarQube 4.0.

X: SonarQube 4.0.

O_2: Observación 2, después de la propuesta SonarQube 4.0.

El procedimiento consiste en determinar en primer lugar una tabla de rango de valores, la cual nos permite ubicar valores cuantitativos de los indicadores, por medio de valores cualitativos expresados en este rango.

Posteriormente realizamos la comparación de valores entre indicadores de acuerdo al diseño de contrastación; esta comparación nos permite finalmente aceptar o rechazar la hipótesis de acuerdo a los estándares científicos de la Estadística.

Tabla 26. Para variables cualitativos:

RANGO SATISFACCIÓN
[0 – 1.5] Nada Ventajoso
[1.6 – 3.2] Poco Ventajoso
[3.3 – 5] Altamente Ventajoso

4.3.2. SUPUESTOS DE LA PRUEBA DE HIPÓTESIS

1. Los datos muéstrales seleccionamos por conveniencia, a una muestra de 16, correspondiente al equipo de software de la consultora Everis, así mismo evaluamos los principales indicadores relevantes para la prueba de hipótesis.

2. La hipótesis nula Ho es la negación de la relación existente entre la variable independiente y la variable dependiente y la hipótesis Ha es la afirmación correspondiente.

3. Se utilizó el 95% del nivel de confiabilidad y 5% del nivel de significancia.

4. Se acepta la hipótesis nula si el Valor Calculado Tc es menor al valor en tabla Tt, caso contrario se rechaza la Ho y se acepta la hipótesis alterna Ha.

4.3.3. CÁLCULO DEL VALOR CRÍTICO Y LA FUNCIÓN PRUEBA

Después de analizar las diferencias entre los indicadores en Pre-test (O_1) y post-test (O_2), se puede concluir que para todos los indicadores hay diferencias significativas y una mejora después de implementar la Tecnología .Net.

Tabla 27. Resumen de resultado de valores

N°	Preguntas	Valor Pre	Valor Post	Dif.
01	¿Cómo califica la funcionalidad del SonarQube 4.0 en la certificación de código Java?	1.8750	4.0625	2.19
02	¿Cómo califica la confiabilidad del SonarQube 4.0 en la certificación de código Java?	1.6875	4.1875	2.50
03	¿Cómo califica la usabilidad del SonarQube 4.0 en la certificación de código Java?	1.6250	4.3750	2.75
04	¿Cómo califica la eficiencia del SonarQube 4.0 en la certificación de código Java?	2.1250	4.3125	2.19
05	¿Cómo califica la eficacia del SonarQube 4.0 en la certificación de código Java?	1.8750	3.9375	2.06
06	¿Cómo califica la portabilidad del SonarQube 4.0 en la certificación de código Java?	2.0625	4.1875	2.13
07	¿Cómo califica los reportes emitidos del SonarQube 4.0 en la certificación de código Java?	1.7500	4.5000	2.75
08	¿Cómo califica la verificación del SonarQube 4.0 en la certificación de código Java?	2.0000	4.1250	2.13
09	¿Cómo califica los procesos del SonarQube 4.0 en la certificación de código Java?	1.8750	4.3125	2.44
10	¿Cómo califica los recursos del SonarQube 4.0 en la certificación de código Java?	1.7500	4.3125	2.56
11	¿Cómo califica la satisfacción del SonarQube 4.0 en la certificación de código Java?	1.9375	4.3750	2.44
TOTAL		**1.87**	**4.24**	**2.38**

Cálculo de la diferencia promedio (D)

$$\overline{D} = \frac{\sum \overline{D}i}{n} = \frac{26.1250}{11} = 2.38$$

Cálculo de la desviación estándar (S)

$$S = \sqrt{\frac{\sum (Di - \overline{D})^2}{n-1}}$$

$$S = \sqrt{\frac{0.63}{10}}$$

$$S = 0.25$$

Hipótesis estática

$Ho = 0_1 - 0_2 >= 0$

$Ha = 0_2 - 0_1 > 0$

Cálculo del valor crítico o valor en tabla de la "T" Student (Tt)

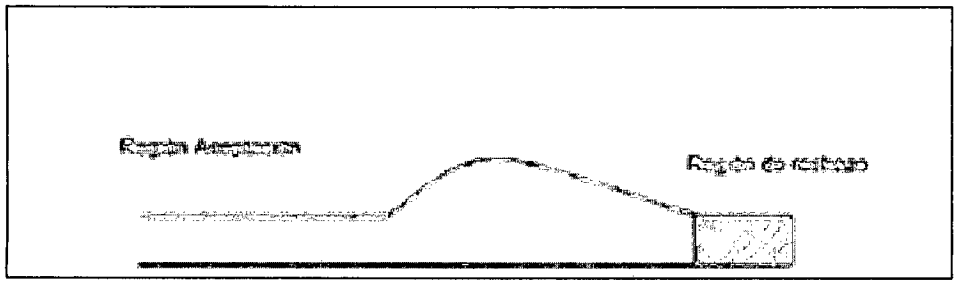
$Tt\ (1 - \square)_{(n-1)}$

$Tt\ (1 - 0.05)_{(11-1)}$

$Tt\ (0.95)_{(10)} = 1.8125$

$Tt = 1.8125$

Regiones de aceptación y rechazo.



Cálculo del valor calculado o Función de prueba (Tc).

$$T_c = \frac{\overline{D}}{\frac{S}{\sqrt{n}}} = \frac{2.38}{\frac{0.25}{\sqrt{11}}}$$

Tc = 2.8466

Por lo tanto:

- Como Tc = 2.8466 es mayor que Tt = 1.8125; entonces se rechazan Ho y se acepta Ha = $0_2 - 0_1 > 0$.

Finalmente se concluye que la hipótesis planteada es aceptada.

En las tablas siguientes apreciamos que existe una significación correlación entre las variables para los nivel del 1% y 5% de manera bilateral. Realizando la prueba de correlaciones: Analizas - Correlaciones – Bivariadas, se obtiene.

Tabla 28. Correlación de datos del Pre – Test

Ítems		P01	P02	P03	P04	P05	P06	P07	P08	P09	P10	P11
P01	Correlación de Pearson	1	,044	,033	,436	,385	,399	,907**	,405	,431	,531*	,471
	Sig. (bilateral)		,871	,902	,091	,141	,125	,000	,120	,095	,034	,065
P02	Correlación de Pearson	,044	1	,325	,191	,396	,367	,092	-,116	,576*	,083	,235
	Sig. (bilateral)	,871		,219	,479	,129	,162	,736	,669	,019	,760	,381
P03	Correlación de Pearson	,033	,325	1	,501*	,434	,426	,070	,132	,337	,189	,099
	Sig. (bilateral)	,902	,219		,048	,093	,100	,798	,626	,202	,484	,715
P04	Correlación de Pearson	,436	,191	,501*	1	,436	,859**	,374	-,101	,374	,338	,258
	Sig. (bilateral)	,091	,479	,048		,091	,000	,154	,709	,154	,200	,334
P05	Correlación de Pearson	,385	,396	,434	,436	1	,303	,480	,203	,546*	,724**	,471
	Sig. (bilateral)	,141	,129	,093	,091		,255	,060	,452	,029	,002	,065
P06	Correlación de Pearson	,399	,367	,426	,859**	,303	1	,428	-,096	,339	,296	,237
	Sig. (bilateral)	,125	,162	,100	,000	,255		,098	,725	,198	,265	,377
P07	Correlación de Pearson	,907**	,092	,070	,374	,480	,428	1	,211	,539*	,704**	,475
	Sig. (bilateral)	,000	,736	,798	,154	,060	,098		,433	,031	,002	,063
P08	Correlación de Pearson	,405	-,116	,132	-,101	,203	-,096	,211	1	-,114	,191	,120
	Sig. (bilateral)	,120	,669	,626	,709	,452	,725	,433		,675	,479	,658
P09	Correlación de Pearson	,431	,576*	,337	,374	,546*	,339	,539*	-,114	1	,379	,256
	Sig. (bilateral)	,095	,019	,202	,154	,029	,198	,031	,675		,148	,339
P10	Correlación de Pearson	,531*	,083	,189	,338	,724**	,296	,704**	,191	,379	1	,429
	Sig. (bilateral)	,034	,760	,484	,200	,002	,265	,002	,479	,148		,097
P11	Correlación de Pearson	,471	,235	,099	,258	,471	,237	,475	,120	,256	,429	1
	Sig. (bilateral)	,065	,381	,715	,334	,065	,377	,063	,658	,339	,097	
	N	16	16	16	16	16	16	16	16	16	16	16
**. La correlación es significativa al nivel 0,01 (bilateral).												
*. La correlación es significante al nivel 0,05 (bilateral).												

Tabla 29. Correlación de datos del Post – Test

Ítems		P01	P02	P03	P04	P05	P06	P07	P08	P09	P10	P11
P01	Correlación de Pearson	1	,556*	,501*	-.037	.378	.301	.284	-.012	.172	,553*	-.122
	Sig. (bilateral)		.025	.048	.891	.149	.258	.287	.963	.524	.026	.651
P02	Correlación de Pearson	,556*	1	,648**	.009	.252	,726**	.351	.077	.134	,569*	-.234
	Sig. (bilateral)	.025		.007	.973	.347	.001	.182	.776	.620	.022	.382
P03	Correlación de Pearson	,501*	,648**	1	.429	.469	,624**	,654**	.374	.484	,540*	.282
	Sig. (bilateral)	.048	.007		.097	.067	.010	.006	.154	.057	.031	.290
P04	Correlación de Pearson	-.037	.009	.429	1	.332	.372	.438	,520*	,541*	-.130	,567*
	Sig. (bilateral)	.891	.973	.097		.209	.155	.090	.039	.031	.631	.022
P05	Correlación de Pearson	.378	.252	.469	.332	1	.397	.068	.375	.406	.417	-.067
	Sig. (bilateral)	.149	.347	.067	.209		.128	.802	.152	.118	.108	.805
P06	Correlación de Pearson	.301	,726**	,624**	.372	.397	1	.405	.267	.318	.315	-.102
	Sig. (bilateral)	.258	.001	.010	.155	.128		.120	.317	.229	.234	.707
P07	Correlación de Pearson	.284	.351	,654**	.438	.068	.405	1	.440	.374	.110	.392
	Sig. (bilateral)	.287	.182	.006	.090	.802	.120		.088	.153	.685	.133
P08	Correlación de Pearson	-.012	.077	.374	,520*	.375	.267	.440	1	.313	.073	.374
	Sig. (bilateral)	.963	.776	.154	.039	.152	.317	.088		.238	.789	.154
P09	Correlación de Pearson	.172	.134	.484	,541*	.406	.318	.374	.313	1	-.309	.484
	Sig. (bilateral)	.524	.620	.057	.031	.118	.229	.153	.238		.244	.057
P10	Correlación de Pearson	,553*	,569*	,540*	-.130	.417	.315	.110	.073	-.309	1	-.151
	Sig. (bilateral)	.026	.022	.031	.631	.108	.234	.685	.789	.244		.576
P11	Correlación de Pearson	-.122	-.234	.282	,567*	-.067	-.102	.392	.374	.484	-.151	1
	Sig. (bilateral)	.651	.382	.290	.022	.805	.707	.133	.154	.057	.576	
	N	16	16	16	16	16	16	16	16	16	16	16
**. La correlación es significativa al nivel 0,01 (bilateral).												
*. La correlación es significante al nivel 0,05 (bilateral).												

CAPITULO V

DISCUSIÓN DE RESULTADOS

5.1. CONSTRASTACIÓN DE RESULTADOS

Sonar es una aplicación que permite analizar distintos parámetros de calidad del software en proyectos informáticos. Con un sistema flexible y adaptable a las necesidades del proyecto, permite controlar varios aspectos del software como la complejidad ciclomática, el grado de replicación de código, la cobertura con tests o el respeto por los coding standards.

Manteniendo estos parámetros bajo control con esta herramienta gratuita conseguiremos un diagnóstico constante del estado de salud de nuestro código, lo que permitirá añadir funcionalidades sobre el mismo de un modo más rápido y por tanto más económico.

Tabla 30. Arquitectura del Sonar

Concepto	Definición	Temas relacionados
Analizador	Procesos batch que analizar el código fuente de obteniendo la imagen instantánea.	Ver información sobre Arquitectura
Cliente	Procesos batch ejecutados por el analizador.	Ver información sobre Arquitectura
Base de datos	Comerciales: Configuración del Sonar instantánea	Ver información sobre Arquitectura
Servidor	Web Server que se utiliza para navegar por instantáneas fotos y hacer la configuración	Ver información sobre Arquitectura

Tabla 31. Calidad del Sonar

Concepto	Definición	Temas relacionados
Verificar	Verificar = Reglas de Codificación.	
Reglas de Codificación	Son una buena práctica de codificación. La evidencia de las reglas de codificación reduce la calidad. Las reglas de codificación pueden comprobar la calidad de los archivos, las pruebas unitarias o paquetes.	Ver Visor de resultados de las pruebas unitarias
Métricas	Una propiedad de un recurso. Ejemplos: número de líneas de código, complejidad, etc. Una métrica puede ser: *cualitativa*: da una indicación de la calidad del recurso (por ejemplo: densidad de las líneas duplicadas, la cobertura de la línea de pruebas unitarias, etc.) o *cuantitativa*: no da una indicación de la calidad del recurso (por ejemplo: número de líneas de código, complejidad, etc.) El valor de la métrica para un recurso dado se denomina medida.	Ver Navegación por las métricas
Medida	El valor de un indicador para un recurso dado en un momento dado. Ejemplos: 125 líneas de código en la clase MyClass Densidad de líneas duplicadas de un 30,5% en el proyecto myProjec	
Perfil de Calidad	Un conjunto de reglas de codificación. Cada instantánea se basa en un perfil de calidad único.	Ver Perfil de Calidad.
Recurso	Una pieza del software. Por ejemplo: proyecto, paquete, modulo, archivo, etc.	
Instantánea	Un conjunto de medidas y evidencias en un recurso determinado en un momento dado. Una instantánea se genera para cada análisis Sonar.	

Evidencia	Cuando un recurso no cumple con una regla de codificación, Sonar registra una evidencia en la instantánea. Una evidencia se puede registrar en un archivo o en una prueba de unidad o en un paquete.	

Tabla 32. Interface web del Sonar

Concepto	Definición	Temas relacionados
Cuadro de mandos	Página Web que proporciona una forma de mostrar los datos almacenados en la base de datos. Un panel es un conjunto de widgets.	Ver Cuadro de mandos
Widget	Es un cuadro que muestra los datos en un cuadro de mandos. Hay dos tipos de widget: *Widget Global* que muestra datos de varios proyectos. Hasta el momento sólo hay un widget global disponible: ver filtros. *Widget de Proyecto* que muestra sólo los datos de un proyecto específico.	

5.1.1. DEFINICIÓN DE MÉTRICAS

Las métricas son el corazón de Sonar, utilizando un sonar de manera eficiente significa comprender perfectamente la definición y el algoritmo de cálculo de cada uno.

En este documento nos centraremos en las métricas para Java, en los siguientes enlaces se presentan las métricas que utiliza Sonar para cada uno de los lenguajes especificados:

- ABAP, C, C++, Cobol, Java, Natural, PL/I, PL/SQL, VB6

5.1.2. DEFINICIÓN DE MÉTRICAS JAVA

Tabla 33. Definición de métricas Java por tamaño

Nombre	Clave	Cualitativa	Descripción
Líneas físicas	lines	no	Número de retornos de carro.

Líneas de comentario	comment_lines	no	Número de javadoc, multi-comentarios y líneas de comentario de una sola línea.
			Las líneas vacías de comentario como, comentarios de cabecera de archivo (utiliza principalmente para definir la licencia) y líneas de código comentadas no están incluidos.
			/**
			* Esto es un bloque de javadoc
			*
			*/
			/*
			* Esto es un bloque de multi-comentario
			*/
			// Esto es un comentario simple
			// log("Debug información"); <- línea de código comentada, no es un comentario

Líneas de código	ncloc	no	Número de líneas físicas de código - número de líneas en blanco - número de líneas de comentario - número de comentarios del archivo de cabecera
Densidad de líneas de comentarios	comment_lines_density	si	= Número de líneas de comentario / (Número de líneas de código + número de líneas de comentario) * 100 Con estas fórmulas: 50% significa que hay el mismo número de líneas de código y líneas de comentarios El 100% significa que el archivo contiene sólo las líneas de comentarios
Paquetes	packages	no	Número de Paquetes
Clases	classes	no	Número de clases como clases anidadas, interfaces, enumeraciones y anotaciones.
Archivos	files	no	Número de archivos analizados.
Directorios	directories	no	Número de directorios analizados.
Accesos	accessors	no	Número de métodos get y set utilizados para poder leer y escribir una propiedad de clase. **Get**

```
1  private String field;
2
3  // Reconocido como un get por Sonar
4  public String getField() {
5    return field;
6  }
7
8  // Reconocido como un get por Sonar
9  public String getMyOwnField() {
10   return field;
11 }
12
13 // No reconocido como un get por Sonar
14 public String getField() {
15   String a = "toto";
16   return field;
17 }
```

Set

```
1  private String field;
2
3  // Reconocido como un set por Sonar
4  public void setField(String field) {
5    this.field = field;
6  }
7
8  // Reconocido como un set por Sonar
9  public void setMyOwnField(String myOwnField) {
10   this.field = myOwnField;
11 }
12
13 // No reconocido como un set por Sonar
14 public void setField(String field) {
15   if (field == null) {
16     this.field = "My field is null";
17   } else {
```

			18 this.field = field; 19 } 20}
Métodos	functions	no	Número de métodos. Nota: Los descriptores de acceso son considerados como métodos si la variable analyse.property.accessors sonar.squid se establece a false. Los constructores son considerados como métodos.
API pública	public_api	no	= Número de clases públicas + número de métodos públicos + número de propiedades públicas (públicas sin los estáticos finales)
API Pública no documentada	public_undocumented_api	si	Número de API pública sin Javadoc.
Densidad de API pública documentada.	public_documented_api_density	si	= (Número de API Pública - Número de API Públicas no documentada) / Número de API pública * 100

Declaraciones	statements	no	Número de declaraciones tal como se definen en la especificación del lenguaje Java, pero sin definiciones de bloque. El contador de declaraciones se incrementa en uno cada vez que se encuentra una expresión, if, else, while, do, for, switch, break, continue, return, throw, synchronized, catch, finally:
			//
			i = 0;
			if (ok)
			if (exit) {
			if (3 == 4);
			if (4 == 4) { ; }
			} else {
			try{}
			while(true){}
			for(...){}
			...
			El contador de declaraciones no se incrementa por una clase, método, campo, definición anotación o por un paquete y declaración de importación.

Tabla 34. Definición de métricas Java por pruebas

Nombre	Clave	Cualitativa	Descripción
Pruebas Unitarias	tests	no	Número de pruebas unitarias.
Duración de las pruebas unitarias	test_execution_time	no	Tiempo para ejecutar las pruebas unitarias
Pruebas unitarias erróneas	test_errors	si	Número de pruebas unitarias que fallaron
Pruebas unitarias falladas	test_failures	si	Número de pruebas de unidad que ha fallado con una excepción inesperada
Densidad de Pruebas unitarias correctas	test_success_density	si	(Pruebas unitarias - (+ Pruebas con errores)) / Pruebas unitarias * 100
Pruebas unitarias omitidas	skipped_tests	si	Number of skipped unit tests
Cobertura de línea	line_coverage	si	La cobertura de línea corresponde con la cantidad de líneas de código que se ejecutan en cada una de las pruebas unitarias realizadas: Cobertura de línea = LC / EL donde LC - Líneas cubiertas (líneas a cubrir – líneas no cubiertas) EL – número total de líneas ejecutables (líneas a cubrir)
Cobertura de línea nueva	new_line_coverage	si	idéntica a la cobertura de línea pero restringido al código fuente nuevo / o actualizado
Cobertura de rama	branch_coverage	si	En cada línea de código que contiene algunas expresiones booleanas, la cobertura de rama responde a la pregunta: "¿Cada expresión booleana es evaluada tanto en true y false?". A nivel de proyecto, esto es la densidad de ramas posibles en las estructuras de control de flujo que se han seguido. Cobertura de rama = (CT + CF) / (2*B) donde CT - ramas que evaluaron a "true" al menos una vez CF - ramas que evaluaron a "false" al menos una vez (CT + CF = conditions_to_cover - uncovered_conditions)

			B – número total de ramas (2*B = conditions_to_cover)
Cobertura de nueva rama	new_branch_coverage	si	idéntica a la cobertura de rama, pero restringido al código fuente nuevo / o actualizado
Cobertura	coverage	si	La métrica de Cobertura es una mezcla de las anteriores; coberturas de línea y coberturas de rama, para obtener una respuesta más precisa a la pregunta "¿cuánto de un código fuente está siendo ejecutado por sus pruebas unitarias?". La cobertura se calcula con la siguiente fórmula: Cobertura = (CT + CF + LC)/(2*B + EL) donde CT - ramas que evaluaron a "true" al menos una vez CF - ramas que evaluaron a "false" al menos una vez LC - Líneas cubiertas (líneas a cubrir – líneas no cubiertas) B - número total de ramas (2*B = conditions_to_cover) EL - número total de líneas ejecutables (líneas a cubrir)
Cobertura nueva	new_coverage	si	idéntica a la cobertura, pero restringido al código fuente nuevo / o actualizado
Condiciones cubiertas	conditions_to_cover	no	Número total de condiciones que podrían ser objeto de pruebas unitarias.
Nuevas condiciones de cobertura	new_conditions_to_cover	no	idénticas a las condiciones cubiertas, pero restringido al código fuente nuevo / o actualizado
Líneas cubiertas	lines_to_cover	no	Número total de líneas de código que pueden ser objeto de pruebas unitarias.
Líneas nuevas cubiertas	new_lines_to_cover	no	idénticas a las líneas cubiertas pero restringido al código fuente nuevo / o actualizado
Condiciones no cubiertas	uncovered_conditions	no	Número total de condiciones que no están cubiertos por las pruebas unitarias.
Nuevas condiciones no cubiertas	new_uncovered_conditions	no	idénticas a las condiciones no cubiertas pero restringido al código fuente nuevo / o actualizado
Líneas no cubiertas	uncovered_lines	no	Número total de líneas de código que no están cubiertos por las pruebas unitarias.
Líneas nuevas no cubiertas	new_uncovered_lines	no	idénticas a las líneas no cubiertas pero restringido al código fuente nuevo / o actualizado

Tabla 35. Definición de métricas Java por duplicidad

Nombre	Clave	Cualitativa	Descripción
Líneas duplicadas	duplicated_lines	si	Número de líneas físicas duplicadas
Bloque duplicados	duplicated_blocks	si	Número de bloques de líneas duplicadas
Archivos con duplicados	duplicated_files	si	El número de archivos que intervienen en la duplicación de las líneas
Densidad de líneas duplicadas	duplicated_lines_densit y	si	Líneas duplicadas / líneas físicas * 100

Tabla 36. Definición de métricas Java por diseño

Nombre	Clave	Cualitativa	Descripción
La profundidad del árbol de herencia	dit	no	La profundidad del árbol de herencia (DIT) proporciona para cada clase una medida de los niveles de herencia superior a la jerarquía de objeto. En Java, donde todas las clases heredan de objeto, el valor mínimo de DIT es 1.
Número de hijos	noc	no	Un número de clase hijos (NOC) simplemente mide el número de descendientes directos e indirectos de la clase.
Respuesta por clase	rfc	no	Ver RFC.
Acoplamiento referente	ca	no	Una clase de acoplamiento referente es una medida de cuántas otras clases utilizar la clase específica
Acoplamiento aferente	ce	no	Una clase de acoplamiento eferente es una medida de cuántas clases diferentes son utilizados por la clase específica.
Falta de cohesión de los métodos	lcom4	si	Ver LCOM4.
Ciclos del paquete	package_cycles	si	Número mínimo de ciclos de detección de paquete para ser capaz de identificar todas las dependencias no deseadas.
Dependencias de paquetes a eliminar	package_feedback_edges	no	Número de dependencias de paquetes para cortar a fin de eliminar todos los ciclos entre los paquetes.
Dependencias de archivos a eliminar	package_tangles	no	Número de dependencias de archivos para cortar a fin de eliminar todos los ciclos entre los paquetes.
Límite de peso de los paquetes	package_edges_weight	no	Número total de dependencias de archivos entre paquetes.

Índice de enredo de los paquetes	package_tangle_index	si	Indica el nivel de enredo de los paquetes, el mejor valor de 0% significa que no hay ciclos y el peor valor del 100% significa que los paquetes son muy enredados. El índice se calcula utilizando: 2 * (package_tangles / package_edges_weight) * 100.
Ciclos de archivos	file_cycles	si	Número mínimo de ciclos de archivos detectados dentro de un paquete para ser capaz de identificar todas las dependencias no deseadas.
Dependencias de archivos sospechosas	file_feedback_edges	no	Dependencias de archivo para cortar con el fin de eliminar los ciclos de entre los archivos dentro de un paquete. Advertencia: Los ciclos no son siempre malos entre los archivos dentro de un paquete.
Archivos enredados	file_tangles	no	file_tangles = file_feedback_edges.
Límite de peso de los archivos	file_edges_weight	no	Número total de dependencias de los archivos dentro de un paquete.
Índice de enredo de los archivos	file_tangle_index	si	2 * (Archivos enredados / Límite de peso de los archivos) * 100.

Tabla 37. Definición de métricas Java por complejidad

Nombre	Clave	Cualitativa	Descripción
Complejidad	complexity	no	El número Complejidad ciclomática es también conocido como McCabe Metric. Simplemente cuenta el número de 'if', 'for', 'while', etc. declarados en un método. Siempre que el flujo de control de un método se divide, el contador ciclomático se incrementa en uno. Cada método tiene un valor mínimo de 1 por defecto excepto el de acceso que no se consideran como método con lo que no incremente la complejidad. Para cada una de las siguientes palabras clave / declaración en Java este valor se incrementa en uno: if for while case catch throw return (que no es la última instrucción de un método);

			&&
			||
			?
			
			Tenga en cuenta que else, default, and finally no incrementan el valor CCN. Por otro lado, un método sencillo con una sentencia switch y switch puede tener un valor CCN sorprendentemente alta (tiene el mismo valor que al convertir a la secuencia equivalente de instrucciones IF).
			Por ejemplo, el siguiente método tiene una complejidad de 5
			
			public void process(Car myCar){ <- +1
			if(myCar.isNotMine()){ <- +1
			return; <- +1
			}
			car.paint("red");
			car.changeWheel();
			while(car.hasGazol() && car.getDriver().isNotStressed()){ <- +2
			car.drive();
			}
			return;
			}

Complejidad promedio por método	function_complexity	si	Número medio de la complejidad ciclomática por método.
Distribución de la complejidad por método	function_complexity_distribution	si	Número de métodos para una complejidad dada.
Complejidad media por clase	class_complexity	si	Complejidad ciclomática promedio por clase
Distribución de la complejidad por clase	class_complexity_distribution	si	Número de clases para una complejidad dada.
Complejidad promedio por archivo	file_complexity	si	Complejidad ciclomática promedio por archivo

Tabla 38. Definición de métricas Java por reglas

Nombre	Clave	Cualitativa	Descripción
Evidencias	violations	si	Número total de evidencias de reglas.
Nuevas evidencias	new_violations	si	Número total de evidencias nuevas.
xxxxx evidencias	xxxxx_violations	si	Número de evidencias con severidad xxxxx, xxxxx bloqueantes, críticas, alta, baja o informativa.
Nuevas xxxxx evidencias	new_xxxxx_violations	si	Número de evidencias nuevas con severidad xxxxx, xxxxx bloqueantes, críticas, alta, baja o informativa.
Ponderación de evidencias	weighted_violations	si	Suma de las evidenciaes ponderada por el coeficiente asociado a cada prioridad (SUM (xxxxx_violations xxxxx_weight *)) Para establecer el peso de cada prioridad, inicie sesión como administrador, vaya a Configuración> General y establecer la propiedad de peso Reglas.
Reglas del índice de cumplimiento (RCI)	violations_density	si	RCI = 100 - (weighted_violations / nloc * 100)

Tabla 39. Definición de métricas Java por SCM

Nombre	Clave	Cualitativa	Descripción
Autores de línea	authors_by_line	no	El último comité en cada linea de código.

5.1.3. ANÁLISIS DE CODIGO FUENTE

Cuando la instalación se haya completado, Sonar está listo para ejecutar el análisis de sus proyectos.

OBJETO DE ANÁLISIS

Sonar tiene la capacidad para realizar análisis en más de 18 lenguajes. El resultado de este análisis va a ser principalmente las medidas de las métricas y las evidencias de las reglas de codificación. Sin embargo, lo que se analizó por Sonar variará dependiendo del lenguaje analizado:

- En todos los lenguajes, Sonar realiza un análisis estático de código fuente (archivos java, programas Cobol, etc.)
- Sonar puede realizar el análisis estático de código compilado en ciertos lenguajes (archivos .class o jars en Java, dll en C#, etc.)
- Sonar puede realizar un análisis dinámico de código en algunos lenguajes (ejecución de pruebas unitarias en Java, C#, etc.)

ANÁLISIS DE EJECUCIÓN

En primer lugar, se debe instalar el lenguaje del proyecto a analizar ya sea por descarga directa o por el centro de actualización.

Para iniciar el análisis con Sonar, están disponibles los siguientes clientes:

- Sonar Runner (cliente recomendado)

- Sonar Ant Task
- Maven
- Motores de CI

Desde Sonar 3.3, es posible analizar un proyecto de varios módulos con diferentes lenguajes. Para las versiones anteriores a la 3.3, cada uno de los módulos tiene que ser analizado por separado mediante la ejecución de un análisis específico. A continuación, los resultados pueden ser agregados a nivel de proyecto, con el plugins de visualización.

ANÁLISIS CON SONAR RUNER

El Sonar Runner se recomienda como cliente por defecto para analizar un proyecto con Sonar.

Requisitos

Es necesario tener instalado el Sonar Runner y haber leído el apartado análisis de código Fuente.

Sonar Runner 2.0+

En este apartado se describe como utilizer el cliente Sonar Runner 2.0+

Utilización

Proyecto Simple

Crear un archivo de configuración en el directorio raíz del proyecto: sonar-project.properties

```
# metadatos necesarios
sonar.projectKey=my:project
sonar.projectName=My project
sonar.projectVersion=1.0

# ruta a los directorios de origen (requerido) sonar.sources=srcDir1,srcDir2

# ruta a los directorios de origen de las pruebas (opcional)
sonar.tests=testDir1,testDir2

# ruta a los binarios del proyecto (opcional), por ejemplo el directorio del código java
sonar.binaries=binDir

# opcional, lista separada por comas de las rutas de acceso a las bibliotecas. Solo se permite la ruta al archive JAR y las rutas a las clases.
sonar.libraries=path/to/library.jar,path/to/classes/dir

# Su valor determina el lenguaje utilizado.
sonar.language=cobol

# Parámetros adicionales
sonar.my.property=value
```

Ejecute el siguiente comando desde el directorio raíz del proyecto para iniciar el análisis Sonar:

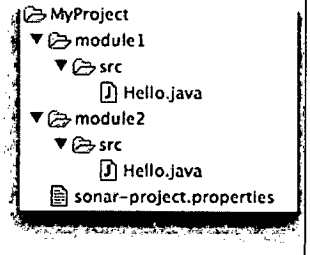
```
sonar-runner
```

Para cada lenguaje, existe un ejemplo de un proyecto simple disponible en github que puede ser consultado o descargado: projects/languages

Proyectos Multi Módulos

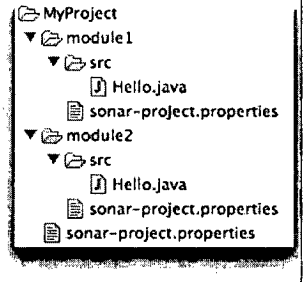
Hay dos maneras de definir una estructura de múltiples módulos en Sonar:

<table>
<tr><th></th><th>Usando la estructura de archivos dado</th><th>... con los archivos 'properties' dados</th></tr>
<tr><td>Caso #1

Establecer toda la configuración en el archivo de propiedades de la carpeta raíz</td><td>
</td><td>Contenido del archivo "MyProject/sonar-project.properties"
<pre>
1 # Información de la raíz del proyecto
2 sonar.projectKey=org.mycompany.myproject
3 sonar.projectName=My Project
4 sonar.projectVersion=1.0-SNAPSHOT
5
6 # Algunas de las propiedades que serán heredados por los módulos
7 sonar.sources=src
8
9 # Lista de los identificadores de módulo
10 sonar.modules=module1,module2
11
12 # Propiedades, que pueden ser anuladas por cada módulo - ejemplo el prefijo con el ID de módulo
13 module1.sonar.projectName=Module 1
14 module2.sonar.projectName=Module 2
15
</pre></td></tr>
</table>

UNIVERSIDAD NACIONAL DE UCAYALI
VICERRECTORADO DE INVESTIGACIÓN
DIRECCIÓN GENERAL DE PRODUCCIÓN INTELECTUAL
OFICINA EJECUTIVA DE BIBLIOTECA Y PUBLICACIONES

<table>
<tr>
<td>Caso #2

Ajuste la configuración de los múltiples archivos de propiedades</td>
<td>
</td>
<td>Contenido del archivo "MyProject/sonar-project.properties"

```
1  # Información de la raíz del proyecto
2  sonar.projectKey=org.mycompany.myproject
3  sonar.projectName=My Project
4  sonar.projectVersion=1.0-SNAPSHOT
5
6  # Algunas de las propiedades que serán heredados por los módulos
7  sonar.sources=src
8
9  # Lista de los identificadores de módulo
10 sonar.modules=module1,module2
```

Contenido del archivo "MyProject/module1/sonar-project.properties"

```
1 # Redefinir las propiedades
2 # No es necesario definir la propiedad
3 sonar.projectName=Module 1
```

Contenido del archivo "MyProject/module2/sonar-project.properties"

```
1 # Redefinir las propiedades
2 # No es necesario definir la propiedad
3 sonar.projectName=Module 2
```
</td>
</tr>
</table>

Cabe destacar:

- Las propiedades se heredan de padres a hijos

  Esto puede ser evitado:

  Mediante el prefijo del identificador del modulo (Caso #1)

Definiéndolos en el archive 'sonar-project.properties' ubicado en el módulo (Caso #2)

- El directorio del modulo base se puede especificar para casos especial

  Por defecto, el directorio del módulo base se especifica desde el identificador de módulo (como en los ejemplos anteriores). Pero puede modificarse usando la variable 'sonar.projectBaseDir'.

  Por ejemplo, aquí hay dos casos de uso y la forma de redefinir el directorio de los módulos base:

  La carpeta de un módulo contiene espacios en blanco o caracteres especiales:

```
module1.sonar.projectBaseDir=My Module One
```

  El módulo no está situado directamente en la carpeta principal, sino en una estructura de directorio más profundo:

```
module1.sonar.projectBaseDir=modules/mod1
module2.sonar.projectBaseDir=modules/mod2
```

- Un proyecto que define los módulos (o el módulo que define sub-módulos) no se puede definir en una carpeta de código fuente para ser analizados por Sonar.

Para cada tipo de estructura, existe un ejemplo disponible en github que puede ser consultado o descargado.

Proyectos Multi-Módulo y Multilenguaje

Desde Sonar 4.0, es posible ejecutar un análisis en un proyecto multi-módulo cuyos módulos contiene el código fuente de varios lenguajes.

Además de la configuración multi-módulo, es obligatorio estableces la propiedad lenguaje para cada módulo:

module.sonar.language

```
1  module1.sonar.language=java
2  module2.sonar.language=javascript
3  ...
```

Se puede consultadar o descargar más información en github.

Uso avanzado

Si el archive sonar-project.properties no puede ser creado en el directorio raíz del proyecto, existen otras alternativas:

- Las propiedades se pueden especificar directamente a través de la línea de comandos:

```
sonar-runner -Dsonar.projectKey=myproject -Dsonar.sources=src1 ..
```

- La variable 'project.settings' puede ser utilizadas para especificar la ruta de acceso al archive de configuración del

proyecto (esta opción es incompatible con la variable 'project.home'):

```
sonar-runner -Dproject.settings=../myproject.properties
```

El directorio de trabajo de Sonar se puede establecer mediante la variable 'sonar.working.directory' (por defecto es '.sonar').

La carpeta raíz del proyecto de análisis se pueden establecer a través de la variable 'project.home' . Esta carpeta debe contener un archivo sonar-project.properties si las variables obligatorias (como 'sonar.projectKey') no se especifican en la línea de comandos.

Otros parámetros de análisis se pueden definir en el archivo de configuración del proyecto a través de parámetros de línea de comandos.

Solución de problemas

Si aparece el error java.lang.OutOfMemoryError es necesario establecer la variable de entorno SONAR_RUNNER_OPTS como se especifica:

```
export SONAR_RUNNER_OPTS="-Xmx512m -XX:MaxPermSize=128m"
```

En entornos Windows evitar las comillas dobles, ya que se enterpreta erróneamente, convirtiendo los dos parámetros en uno solo:

```
set SONAR_RUNNER_OPTS=-Xmx512m -XX:MaxPermSize=128m
```

Migrando de Sonar Runner 1.x a Sonar Runner 2.0

Cambie las siguientes variables en el archivo de sonar-project.properties:

- sources => sonar.sources
- tests => sonar.tests
- binaries => sonar.binaries
- libraries => sonar.libraries

Configura explícitamente la variable sonar.sourceEncoding en el archivo de sonar project.properties.

ANÁLISIS CON MAVEN

Este cliente está recomendado para realizar el análisis de proyectos de Java Maven.

Requisitos

Es necesario tener instalado y configurado el Sonar Maven y haber leído el apartado análisis de código Fuente.

Analizando un proyecto Maven

El análisis de un proyecto Maven consiste en ejecutar el cliente de Maven en el directorio donde se encuentra el archivo pom.xml.

Forma recomendada

```
mvn clean install -DskipTests=true
mvn sonar:sonar
```

skipTests=true permite no ejecutar las pruebas unitarias en dos ocasiones: durante la instalación y la inclusión del objeto en Sonar. Tambien se puede desactivar la ejecución de las pruebas integradas. Revisar la documentación del Maven.

Usando Eclipse

Asegúrese de que no está utilizando el plugin de Maven embebido en el eclipse. Utilice la última versión del plugin de Maven para eclipse y desmarque la opción "resolver los artefactos del espacio de trabajo" en la ventana de lanzamiento del proyecto Maven. Echa un vistazo a el primer comentario de esta entrada:

http://jira.codehaus.org/browse/SONAR-929

Opciones avanzadas

Tenga en cuenta que las opciones avanzadas (por ejemplo, "- projects" y "--resume-from") no son compatibles con Sonar y no debe ser utilizado.

Pasos alternativos

Cuando no es posible utilizar la configuración anterior, las pruebas unitarias se ejecutarán dos veces. No utilice el parámetro DskipTests=true ya que no se ejecutaran las pruebas.

```
mvn clean install sonar:sonar -Dmaven.test.failure.ignore=true
```

Si la variable Dmaven.test.failure.ignore=true asegura que aunque fallen las pruebas unitarias, el análisis del Sonar se realizar.

Configuración del análisis con Sonar

Un ejemplo de archivo pom.xml está disponible aquí.

Parámetros de análisis adicionales están listados en la página de análisis de parámetros.

Véase también algunos ejemplos completos aquí.

Cómo determinar la versión del Plugin de Maven

Es recomendable bloquear la versión del plugin de Maven. Solamente existen dos versiones del plugin de Sonar Maven, uno para Maven 2 y otro para Maven 3. La versión depende de la versión del Maven que se utilice en el proyecto.

Proyectos Analizados con Maven 2 solamente

Agregue el código siguiente al archivo pom.xml:

```
<build>
  <pluginManagement>
        <plugins>
              <plugin>
                    <groupId>org.codehaus.mojo</groupId>
                    <artifactId>sonar-maven-plugin</artifactId>
                    <version>1.0</version>
              </plugin>
        <plugins>
  </pluginManagement>
</build>
```

⚠ La versión 1.0-beta-1 no está soportada.

Proyectos analizados con Maven 3 solamente.

```
<build>
  <pluginManagement>
        <plugins>
              <plugin>
                    <groupId>org.codehaus.mojo</groupId>
                    <artifactId>sonar-maven-plugin</artifactId>
                    <version>2.0</version>
              </plugin>
        <plugins>
  </pluginManagement>
</build>
```

Proyectos analizados con Maven 2 Y Maven 3

```
<build>
   <pluginManagement>
         <plugins>
               <plugin>
                  <groupId>org.codehaus.mojo</groupId>
                  <artifactId>sonar-maven-plugin</artifactId>
                  <version>${sonarVersion}</version>
               </plugin>
         <plugins>
   </pluginManagement>
</build>

<profile>
   <id>maven-2</id>
   <activation>
         <file>
               <!-- basedir expresión que solamente es
               reconocida por el Maven 3.x (ver MNG-2363) -->
               <missing>${basedir}</missing>
         </file>
   </activation>
   <properties>
         <sonarVersion>1.0</sonarVersion>
    </properties>
</profile>
<profile>
   <id>maven-3</id>
   <activation>
         <file>
            <!-- basedir expresión que solamente es reconocida
            por el Maven 3.x (ver MNG-2363) -->
            <exists>${basedir}</exists>
         </file>
   </activation>
   <properties>
         <sonarVersion>2.0</sonarVersion>
   </properties>
</profile>
```

Analizando un proyecto Multi Módulo y Multi Lenguaje

Desde Sonar 3.3, es posible ejecutar un análisis en un proyecto multi-módulo cuyos módulos contiene el código fuente de varios idiomas.

Para ello, sólo tiene que añadir la variable "sonar.language" al pom de cada módulo.

Se puede consultar o descargar más información en github.

ANÁLISIS CON SONAR ANT TASK

En este apartado se explica como integrar Sonar en el Apache Ant.

Requisitos

Es necesario tener instalado y configurado el Sonar Ant Task y haber leído el apartado análisis de código Fuente

Sonar Ant Task 2.0+

En este apartado se describe como usar el Sonar Ant Task 2.0

Utilización

Proyecto simple

Definir un nuevo objeto Sonar Ant en su script de creación:

build.xml

```
...
<!-- Definición de las variables globales de Sonar -->
<property name="sonar.jdbc.url"
value="jdbc:mysql://localhost:3306/sonar?useUnicode=true&amp;characterEncoding=utf8" />
```

```
<property name="sonar.jdbc.username" value="sonar" />
<property name="sonar.jdbc.password" value="sonar" />
...

<!-- Definición de las variables de proyecto de Sonar -->
<property name="sonar.projectKey" value="org.codehaus.sonar:example-java-ant" />
<property name="sonar.projectName" value="Proyecto Simple de Java Project analizado con Sonar Ant Task" />
<property name="sonar.projectVersion" value="1.0" />
<property name="sonar.language" value="java" />
<property name="sonar.sources" value="src" />
<property name="sonar.binaries" value="build" />
...

<!-- Definición del objeto Sonar -->
<target name="sonar">
  <taskdef uri="antlib:org.sonar.ant" resource="org/sonar/ant/antlib.xml">
    <!-- Actualizar la siguiente línea, o poner el archive "sonar-ant-task-*.jar" en el directorio "$HOME/.ant/lib" -->
    <classpath path="path/to/sonar/ant/task/lib/sonar-ant-task-*.jar" />
  </taskdef>

  <!-- Ejecutar Sonar -->
  <sonar:sonar />
</target>

...
```

Ejecute el siguiente comando desde el directorio raíz del proyecto para iniciar el análisis Sonar:

```
ant sonar
```

Se puede consultar o descargar más información en github.

Proyecto Multi Módulo

La configuración Sonar para el raíz y los módulos se debe hacer en el archivo build.xml raíz:

Raíz build.xml

```
...
<!-- Establecer los IDs de los módulos -->
<property name="sonar.modules" value="module-one,module-two"/>

<!-- Para los módulos, las propiedades se heredan de los padres. Pueden sustituirse como se muestra a continuación: -->
<property name="module-one.sonar.projectName" value="Module One" />
<property name="module-one.sonar.sources" value="sources/java" />
<property name="module-one.sonar.binaries" value="target" />

<!-- Por defecto, el directorio base de los módulos es <curent_directory>/<module_ID>. Se puede modificar si es necesario -->
<property name="module-one.sonar.projectBaseDir" value="Module 1" />
...
```

Se puede consultar o descargar más información en github.

Proyectos Multi-Módulo y Multi-Languaje

Desde Sonar 3.3, es posible ejecutar un análisis en un proyecto multi-módulo cuyos módulos contiene el código fuente de varios lenguajes.

Además de la configuración multi-módulo, es obligatorio estableces la propiedad lenguaje para cada módulo:

module.sonar.language

```
1<property name="module1.sonar.language" value="java"/>
2<property name="module2.sonar.language" value="js"/>
3...
```

Se puede consultar o descargar más información en github.

Uso avanzado

Se pueden definir parámetros adicionales para en análisis en el archivo build.xml o mediante línea de comandos.

Ejecutar pruebas y obtener la cobertura de código con Jacoco Ant Task

Se puede consultar o descargar más información en github.

- Proyecto Simple: projects/code-coverage/ut/ant/ut-ant-jacoco-runTests
- Proyectos Multi módulos: projects/code-coverage/ut/ant/ut-ant-jacoco-modules-runTests

Limitaciones conocidas

Las fuentes principales y las fuentes de prueba no se pueden mezclar en el mismo directorio.

Migración de Sonar Ant Task 1.x a Sonar Ant Task 2.0

El Sonar Ant Task 2.0 puede ejecutar el análisis con los archivos de configuración diseñados para Sonar Ant Task 1.X.

Este modo de compatibilidad se activa automáticamente. Pero es recomendable migrar al nuevo formato, ya que el modo de compatibilidad se dará de baja en las próximas versiones.

INTEGRACIÓN CONTINUA

Se recomienda realizar el análisis de proyectos sobre un proyecto estable para obtener lo mejor de la plataforma de Sonar. Servidores de integración continua son las herramientas perfectas para ello.

Hudson / Jenkins

Instalar y configurar el plugin de Sonar para Hudson / Jenkins.

Atlassian Bamboo

Instalar el plugin de Sonar para Bamboo, de Marvelution.

Apache Continuum 1.2

La dificultad de Continuum es ser capaz de sincronizar la instalación con el objeto de sonar porque es necesario que un comando de instalación es ejecutado antes de que el sonar comience a ejecutarse (Ver Análisis de código fuente para más información).

Desde Continuum no se proporciona el mecanismo para desencadenar la acumulación de ejecuciones, para ello hay dos opciones: o la ejecución de comandos de instalación periódicamente (por ejemplo cada hora), o lanzar un comando completo, como:

```
mvn clean install sonar:sonar -Dmaven.test.failure.ignore=true
```

en este caso, las pruebas unitarias se van a ejecutar dos veces: una durante la instalación y otra vez con el sonar.

En caso de que decida ejecutar comandos de instalación de forma regular, aquí está la configuración que puede hacer para el sonar:

- Crea una variable de configuración con el objeto:

```
sonar:sonar
```

Para proyectos multi módulos, asegurar que la opción –no recursivo no está establecida en los argumentos de la definición del build.

- Una vez creado, ejecute la compilación en el directorio raíz del proyecto que quiere analizar con Sonar.

- En el caso de proyectos multi módulos, asegurarse de revisar todos los módulos del proyecto, cargue el root recursivamente cuando se cree un proyecto con el asistente Continuum Maven 2.

CruiseControl

En el CruiseControl, utilice el signo | para separar la instalación y la ejecución del Sonar en dos procesos diferentes.

El comando de lanzamiento queraría de la siguiente forma:

```
clean install -Dtest=false -DfailIfNoTests=false | sonar:sonar -Psonar_profile
```

Jetbrains TeamCity

Estos son los pasos de configuración que se tiene que hacer para la creación:

1. En la configuración general, asegurarse de que al menos una pruebas está marcada antes de la ejecución.
2. En la configuración del ejecutor, seleccione el objeto Maven 2
3. clean install sonar:sonar

   Parámetros Maven adicionales por línea de comandos:

   Dmaven.test.failure.ignore=true
4. En la configuración de la activación seleccione un programa diario (o el necesitado)
5. En las propiedades y variables de entorno, añadir una variable de entorno llamada MAVEN_OPTS, y establecer el valor a '-Xmx512m'

PARÁMETROS DE ANÁLISIS

Los parámetros de configuración del análisis de un proyecto se pueden especificar en varios archivos. Esta es la jerarquía de parámetros:

- Parámetros de análisis globales, definidos en la interfaz de usuario, se aplicará a todos los proyectos.
- Parámetros de análisis de proyectos, definidos en la interfaz de usuario, anulará los parámetros globales.

- Parámetros de análisis del proyecto, definidos en el archivo de configuración del proyecto de análisis o en el archivo de configuración del analizador, se sobreescriben loss definidos en la interfaz de usuario.
- Parámetros de línea de comandos, que se definen en el lanzamiento de un análisis, se anulará cualquier parámetro.

Parámetros obligatorios

Tabla 40. Servidor Sonar

Clave	Descripción	Valor por defecto
sonar.host.url	URL del Servidor Sonar	http://localhost:9000

Tabla 41. Base de datos

Clave	Descripción	Valor por defecto
sonar.jdbc.driverClassName	Controlador JDBC utilizado por Sonar	org.h2.Driver Antes de Sonar 3.2: org.apache.derby.jdbc.ClientDriver
sonar.jdbc.url	URL del controlador JDBC	jdbc:h2:tcp://localhost:9092/sonar Antes de Sonar 3.2: jdbc:derby://localhost:1527/sonar
sonar.jdbc.username	Usuario del controlador JDBC	sonar
sonar.jdbc.password	Contraseña del controlador JDBC	sonar

Tabla 42. Configuración del proyecto

Clave	Descripción	Valor por defecto	Versión
sonar.projectKey	Clave del proyecto, es única para cada proyecto. Establecer mediante <groupId>:<artifactId> al utilizer Maven.		
sonar.projectName	Nombre del proyecto que se mostrará en el interfaz web. Establecer mediante <name> al utilizer Maven.		
sonar.projectVersion	Versión del proyecto. Establecer mediante <version> al utilizer Maven.		

sonar.language	Establece el lenguaje del código Fuente. Si mediante un plugin de Sonar se permite analizar otro lenguaje, el analizador de código fuente tiene que ser activado mediante esta variable. El lenguaje predeterminado se puede configurar a nivel de isntancia desde: Cofiguración > Configuración General > General y modificar la variable sonar.language	java	desde 2.0
sonar.sources	Separados por comas, rutas de acceso a los directorios que contienen los Fuentes. No es compatible con Maven, los fuentes se recuperan del directorio predeterminado para los proyectos Maven Java.		

Parámetros opcionales

Tabla 43. Configuración de proyecto

Clave	Descripción	Valor por defecto	Versión
sonar.projectDescription	Descripción del proyecto. Establecer mediante <description> al usar Maven.		
sonar.binaries	Separados por comas, rutas de acceso a los directorios que contienen los archivos binarios (en el caso de java, los directorios donde se encuentran los archivos .class) No es compatible con Maven, los binarios se recuperan del directorio por defecto establecido para los proyectos Maven Java.		
sonar.tests	Separados por comas, ruta a los directorios que contienen las pruebas. No es compatible con Maven, las pruebas se recuperan del directorio predeternimado en los proyectos Java Maven.		
sonar.libraries	Separados por comas, ruta a los archivos con librerías externas (en el caso de Java, los archivos JAR)		
sonar.sourceEncoding	Codificación de los archivos de origen. Por ejemplo: UTF-8, MacRoman, Shift_JIS. Esta variable puede ser reemplazada	sistema de codificación	

	pro la variable *project.build.sourceEncoding* en proyectos Maven. La lista de las codifdificaciones disponibles depende de la JVM. Ver http://docs.oracle.com/javase/1.5.0/docs/guide/intl/encoding.doc.html.		
sonar.importSources	A veces, por razones de seguridad u otras, fuentes del proyecto no deben ser almacenados y mostrados.	true	desde 1.5
sonar.projectDate	Si es necesario introducir datos históricos y destacar algunos eventos, esto es posible mediante la etiqueta de versión y la opción sonar.projectDate. El formato es yyyy-MM-dd, por ejemplo 2010-12-25.	Fecha actual	desde 1.5
sonar.exclusions	Excluir archivos de análisis. Esta propiedad se suele fijar en la configuración de página del proyecto. Es una lista separada por comas de los patrones de exclusión. Las rutas se definen en el directorio base de origen. Ejemplo:. Com / mycompany / * java, ** / * Dummy.java. La documentación completa de la interfaz web Sonar: ir a miproyecto> Exclusiones.		desde 1.8
sonar.skippedModules	Algunos módulos del proyecto no deben ser analizados y consolidados con las medidas globales del proyecto, por ejemplo, las muestras de pruebas de integración o el código generado. El formato es una lista separada por comas de los módulos: module1_toexclude, module2_toexclude.		desde 1.5
sonar.includedModules	Lista separada por comas de los módulos para el análisis, todos los demás módulos son automáticamente ignorados. Tenga cuidado: la raíz del proyecto se debe agregar a la lista.		desde 2.2
sonar.branch	Administrar ramas SMC. Dos ramas de un mismo proyecto se consideran como diferentes proyectos en Sonar.		desde 1.10

sonar.profile	A través de la interfaz web de Sónar, se puede definir los perfiles de calidad que se desee y se puede asociar un perfil de calidad a un determinado proyecto. También puede hacer esta asociación mediante la variable "sonar.profile".	Perfil predeterminado del lenguaje en cuestión	desde 1.6
sonar.skipDesign	Desactivar el análisis de bytecode de Java. Desde Sonar 2,0, el código de binario de Java es analizada por Sonar con el fin de extraer las dependencias entre los paquetes y archivos. Esas dependencias se utilizan por ejemplo para mostrar el DSM (dependencia del sistema Matriz). Este análisis de bytecode se puede desactivar.	false	desde 2.0
sonar.phase	Run maven phase or goal prior to analysis. When Sonar needs a phase or maven goal to be executed prior to analysis, this parameter can be used. For example sonar.phase=generate-sources. This property is used only for Maven analysis.		desde 1.10
sonar.java.source	Versión del código fuente de Java. Esta propiedad no se utiliza por el núcleo Sonar pero puede ser utilizado por el plugin Sonar Java como el plugin PMD.	1.4, 1.5, 1.6 desde 2.10. 1.7 desde 2.12	desde 2.6
sonar.java.target	Versión del Objeto del código Java. Esta propiedad no se utiliza por el núcleo Sonar pero puede ser utilizado por el plugin Sonar como el plugin de Java Clover.	1.4, 1.5 desde 2.10	desde 2.6
sonar.findbugs.excludesFilters	Permite utilizar FindBugs, excluye los filtros.		desde 2.10

Tabla 44. Unit And Integration Tests

Clave	Descripción	Valor por defecto	Versión
sonar.dynamicAnalysis	El análisis dinámico de las pruebas unitarias. Por defecto, las pruebas unitarias se ejecutan pero si lo desea, puede optar por hacer sólo análisis	true	desde 1.7

	estático o volver a utilizar los informes existentes que han sido previamente generadas. Los valores posibles son verdaderas, falsas, reuseReports.		
sonar.surefire.reportsPath	La ruta absoluta o relativa del directorio de informes. Sólo se utiliza si es sonar.dynamicAnalysis reuseReports.		desde 1.7
sonar.jacoco.reportPath	La ruta absoluta o relativa del archivo de informe jacoco. Sólo se utiliza si es sonar.dynamicAnalysis reuseReports.		desde 2.12
sonar.cobertura.reportPath	La ruta absoluta o relativa del archivo de informe Cobertura xml. Sólo se utiliza si es sonar.dynamicAnalysis reuseReports.		sin desde ce 1.7
sonar.clover.reportPath	La ruta absoluta o relativa del archivo de informe xml. Sólo se utiliza si es sonar.dynamicAnalysis		desde 1.7
sonar.jacoco.itReportPath	La ruta absoluta o relativa del archivo de informe jacoco para las pruebas de integración.		desde 2.12

Tabla 45. Sonar Configuration

Clave	Descripción	Valor por defecto	Versión
sonar.host.connectTimeoutMs and sonar.host.readTimeoutMs	El aumento de los tiempos de espera HTTP de las peticiones al servidor Sonar. El plugin de Maven ejecuta algunas peticiones HTTP al servidor Sonar. Dos tiempos de espera hace que la llamada fallará si la conexión del servidor es demasiado lento. En tal caso, los tiempos de espera se pueden aumentar de propiedades Maven.	respectivamente 30 y 60 milisegundos	desde 1.12

Tabla 46. Log

Clave	Descripción	Valor por defecto	Versión
sonar.showSql	Mostrar todas las solicitudes SQL ejecutadas en modo batch	false	desde Sonar 2.14, Sonar Ant Task 1.4 & Sonar Runner 1.3
sonar.showSqlResults	Mostrar el resultado de todas las peticiones SQL ejecutadas en modo batch	false	desde Sonar 2.14, Sonar Ant Task 1.4 & Sonar Runner 1.3
sonar.verbose	La activación del modo DEBUG de los procesos en modo batch	false	desde Sonar 2.12, Sonar Ant Task 1.3 & Sonar Runner 1.2

5.1.4. EL ESPACIO DE TRABAJO

La herramienta web se divide en tres áreas principales:

- El área de "Home" (página de inicio)
- El área de proyecto (cuadros de mando y el resto de opciones)
- El área de configuración (configuración global y por proyecto)

LA BARRA SUPERIOR

En cualquier lugar de la herramienta, estés donde estés, siempre está la barra superior:



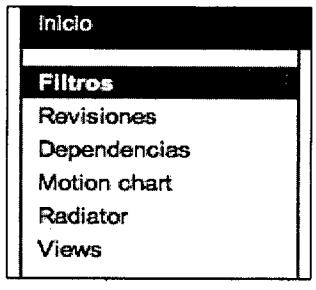
Esta barra incluye diferentes opciones que te permiten realizar tareas generales como por ejemplo:

- Volver a la página de inicio.
- Buscar un proyecto en el campo de búsqueda.
- Ver la miga de pan para saber dónde te encuentras actualmente, y navegar a través de ella.
- Conectarse o desconectarse a la herramienta como usuario o administrador.
- Imprimir la página actual.
- Un enlace permanente a la página.

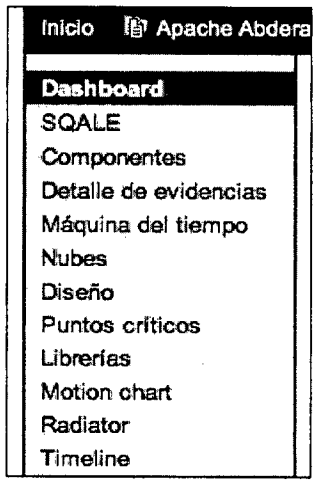
MENÚ LATERAL

El menú lateral es un menú contextual y sus opciones dependen del área en la que te encuentres y de las extensiones que se hayan incorporado en la instalación:

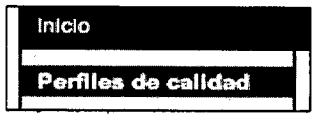
Menú lateral de la página de inicio



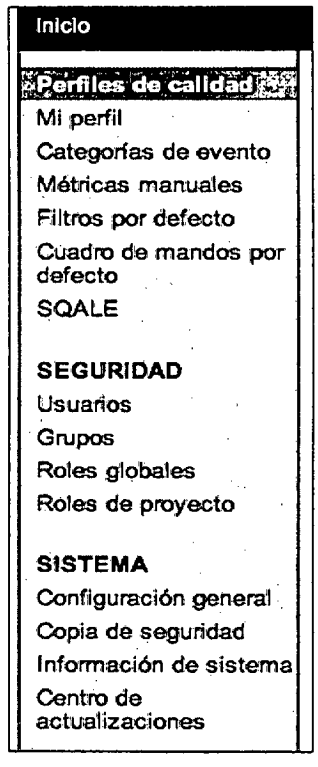
Menú lateral de la página de proyecto



Menú lateral de configuración global del usuario



Menú lateral de configuración global del administrador



Menú lateral de configuración de proyecto



5.1.5. LA PÁGINA DE INICIO

La página de inicio es la página principal de la herramienta una vez nos hemos identificado. Esta página es el punto de partida de la navegación por todos los datos y presenta una visión general con

el listado de proyectos analizados ordenados según las métricas disponibles.

Cada usuario solo podrá visualizar aquellos proyectos sobre los que tenga permisos suficientes. La gestión de permisos se describe en el apartado correspondiente a seguridad.

El principal objetivo de la página de inicio es responder al máximo número de preguntas si necesidad de acceder a páginas más detalladas.

Por ejemplo:

Listado de mis proyectos favoritos

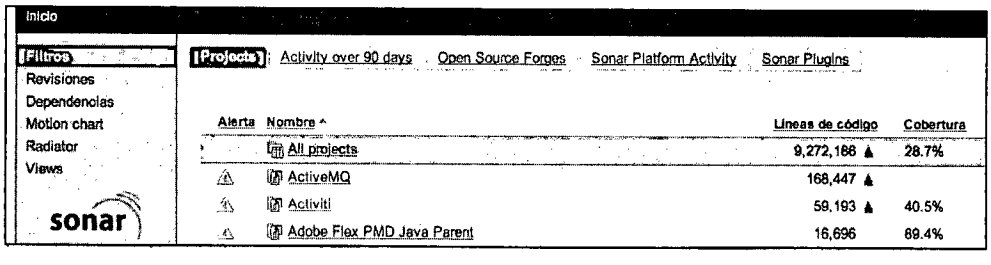
Listado de los peores proyectos desde un criterio específico: deuda técnica, cobertura de código, duplicados,...

Listado de los módulos de un proyecto

Listado de los ficheros más complejos

Listado de las pruebas unitarias que más tardan

Todos estos listados se configuran a través de filtros.



En la página de inicio se pueden ver ciertos indicadores que nos informan si los proyectos cumplen con los requisitos establecidos (umbrales):

	Indica que el proyecto cumple con todos los umbrales establecidos y no hay ninguna alerta a destacar
	Indica que el proyecto ha superado alguno de los umbrales de advertencia en alguna de sus métricas
	Indica que el proyecto ha superado alguno de los umbrales críticos en alguna de sus métricas

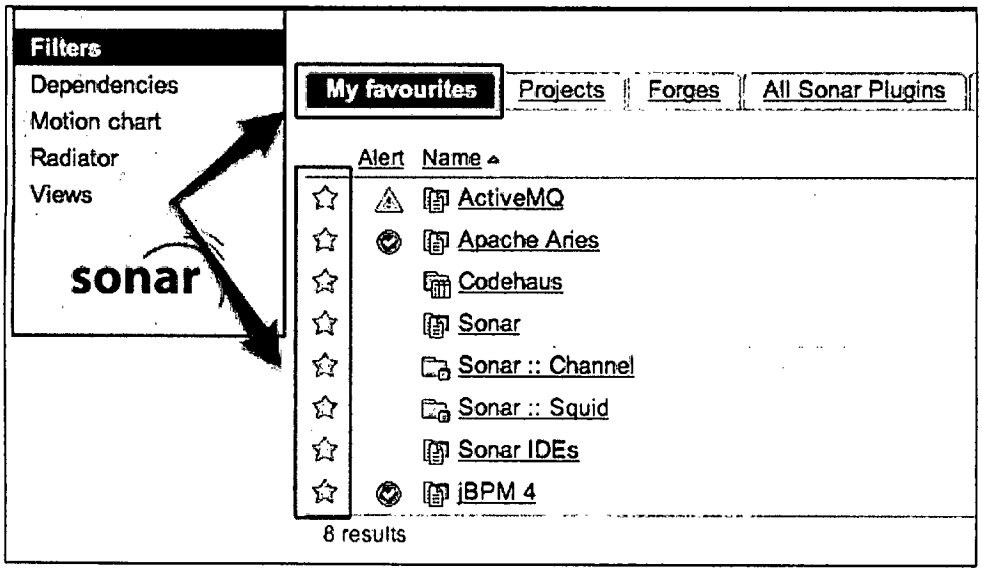
Si en el listado se muestra alguna de las métricas que han superado los umbrales, entonces se representarán coloreadas según el nivel de gravedad: en naranja si se trata de una advertencia, en rojo si se trata de algo crítico. La definición de umbrales se trata en su correspondiente apartado.

Además, se puede observar de forma visual las tendencias que han tenido en los diferentes indicadores: flecha en verde si ha mejorado la tendencia es a mejor, y flecha en rojo si la tendencia es a peor.

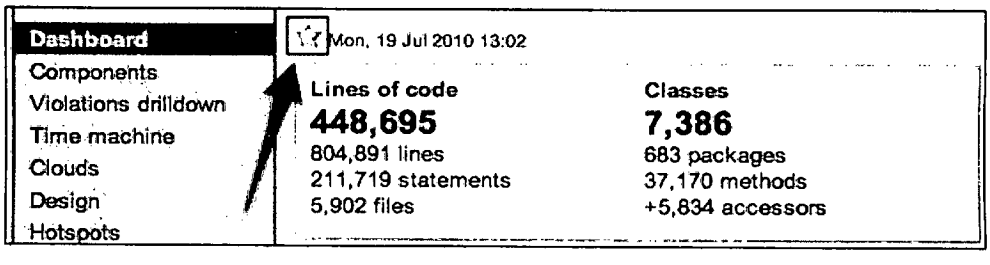
Desde la página de inicio se puede acceder a cualquiera de los recursos que se muestran para navegar en los resultados detallados.

LA PESTAÑA "MIS FAVORITOS"

Esta pestaña muestra todos los recursos, ya sean proyectos, módulos, paquetes o ficheros, que hayan sido marcados como favoritos por el usuario.



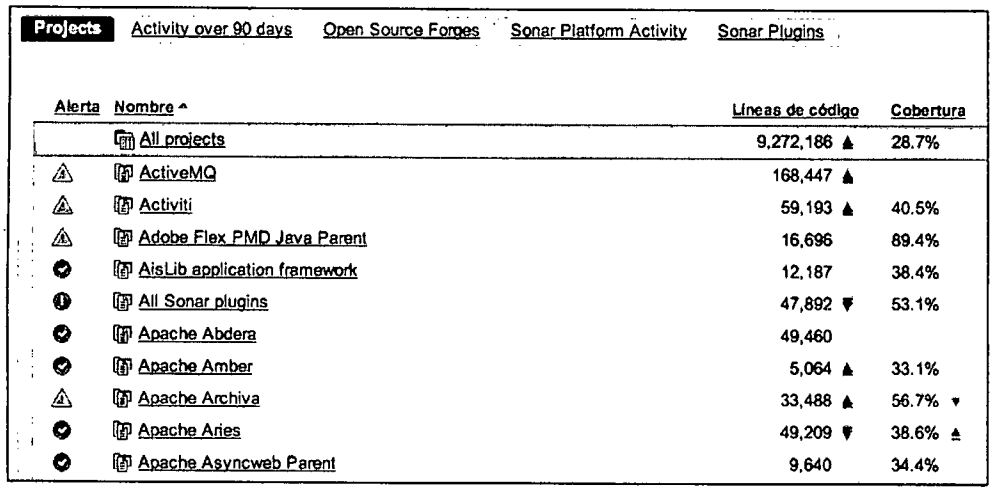
Cualquier recurso puede marcarse como favorito simplemente pinchando en el icono de la estrella disponible en los cuadros de mando, en los listados de proyectos o en las páginas de componentes:



LA PESTAÑA "TODOS LOS PROYECTOS"

Esta pestaña esta disponible para todos los usuarios de la herramienta. Tal y como se espera, muestra todos los proyectos disponibles a los que tenga acceso el usuario.

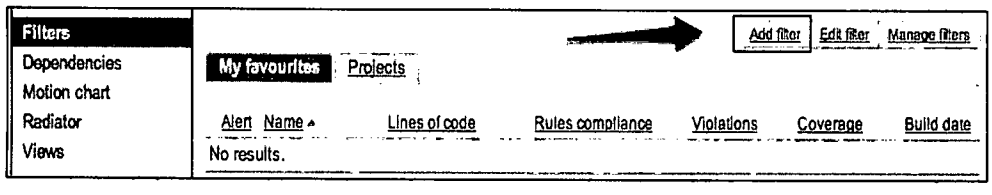
Como en todas las pestañas, pinchando en algunas de las métricas se puede ordenar el listado de forma ascendente o descendente.



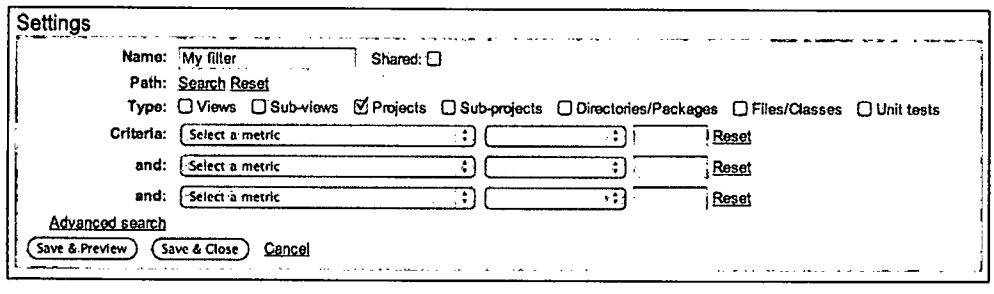
CREACIÓN DE FILTROS

Para la creación de un filtro es necesario que el usuario esté identificado en la herramienta.

Para crearlo basta con pinchar en el botón de "Añadir filtro" en la parte superior derecha de la página de inicio.



A continuación se tiene que completar el formulario de creación del filtro:



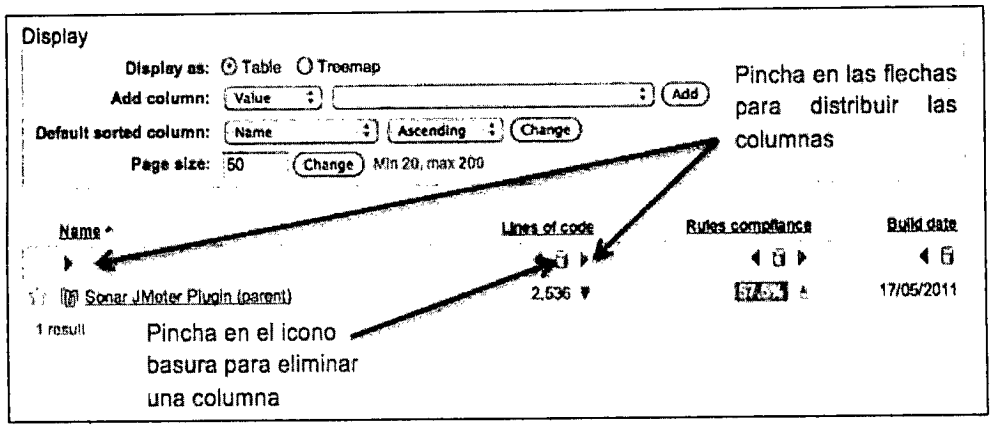
Donde los parámetros disponibles tienen el siguiente significado:

Nombre del campo	Descripción	Obligatorio	Avanzado
Name	El nombre del filtro	✓	
Shared	Esta opción sólo está disponible con el rol de administrador. Permite compartir los filtros con cualquier otro usuario.		
Path	Mecanismo para filtrar solo los recursos que compartan una misma raíz		
Type	Mecanismo para filtrar solo los recursos que sean de un tipo específico		
Criteria	Mecanismo para filtrar solo los recursos que en los cuales una métrica específica es mayor o menor que un umbral.		
Language	Mecanismo para filtrar recursos de un lenguaje concreto		✓
Favourites only	Mecanismo para filtrar solo recursos favoritos		✓
Resource key like	Mecanismo para filtrar solo recursos cuyas claves coincidan con un patrón determinado		✓
Resource name like	Mecanismo para filtrar solo recursos cuyos nombres coincidan con un patrón determinado		✓
Build date	Mecanismo para filtrar solo recursos de acuerdo a la última fecha del análisis		✓

Una vez definido el filtro con lo que se va a mostrar, tendremos dos opciones disponibles:

"*Guardar y cerrar*" (Save & Close): que permite guardar el filtro actual y volver a la página de inicio.

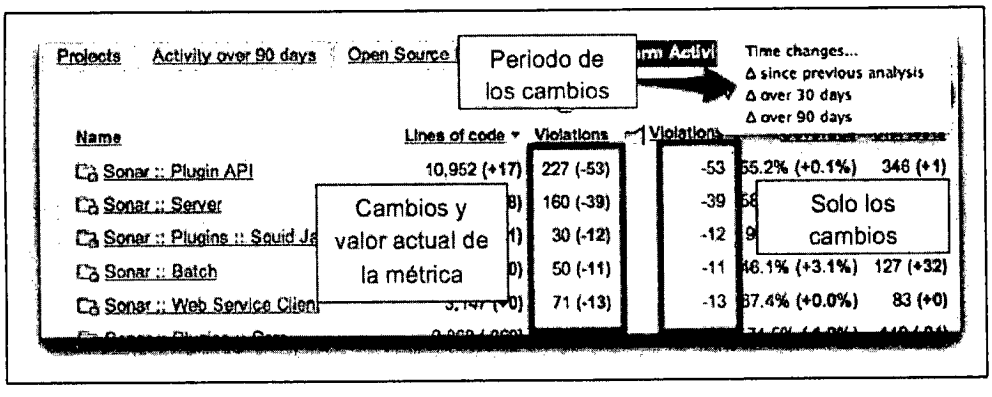
"*Guardar y visualizar*" (Save & Preview): que permite guardar el filtro actual pero permanecer en modo edición, para personalizar los resultados y empezar a ver el resultado del filtro.



FILTROS DIFERENCIALES

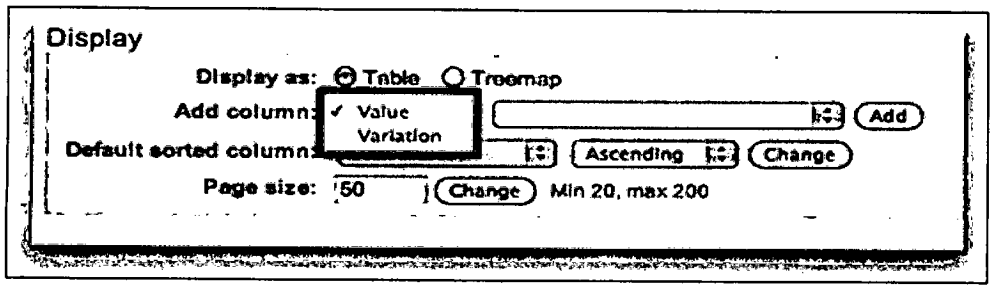
Mediante la utilización de los filtros diferenciales se pueden ver de forma sencilla los cambios en las métricas de un proyecto en periodos de tiempo predefinidos.

Esto es muy importante para poder establecer un ciclo de mejora continua, ya que permite saber exactamente las métricas que han mejorado o empeorado en un periodo de tiempo concreto y configurable.

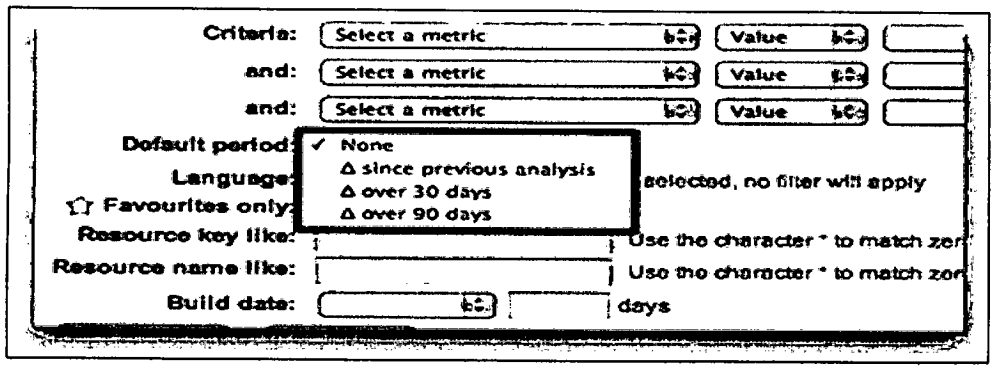


Para poder utilizar esta funcionalidad basta con elegir el periodo de los cambios en la lista desplegable de "Time changes".

Y para crear un filtro especifico que nos permita tener en la página de inicio un listado con los últimos cambios basta con elegir en el filtro las columnas correspondientes, seleccionando si se quiere el valor o la variación.

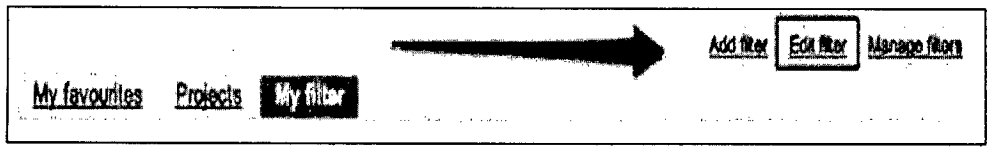


Y posteriormente configurando el periodo de tiempo concreto sobre el que obtener los cambios.



EDITAR UN FILTRO

Cualquier filtro que hayas creado previamente puede ser editado a través del botón "Editar filtro" (Edit filter):



ADMINISTRAR LOS FILTROS

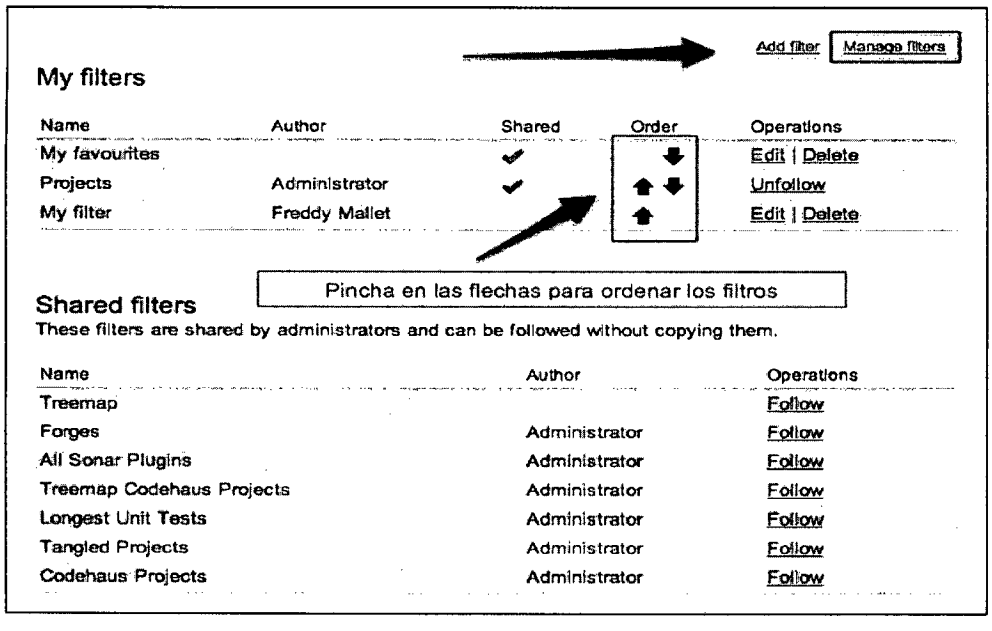
Una vez tengas creados unos cuantos filtros, probablemente sea necesario:

Ordenar esos filtros en la interfaz

Eliminar o editar algunos de ellos

Acceder a filtros compartidos (y seguirlos, para incorporarlos a tú página)

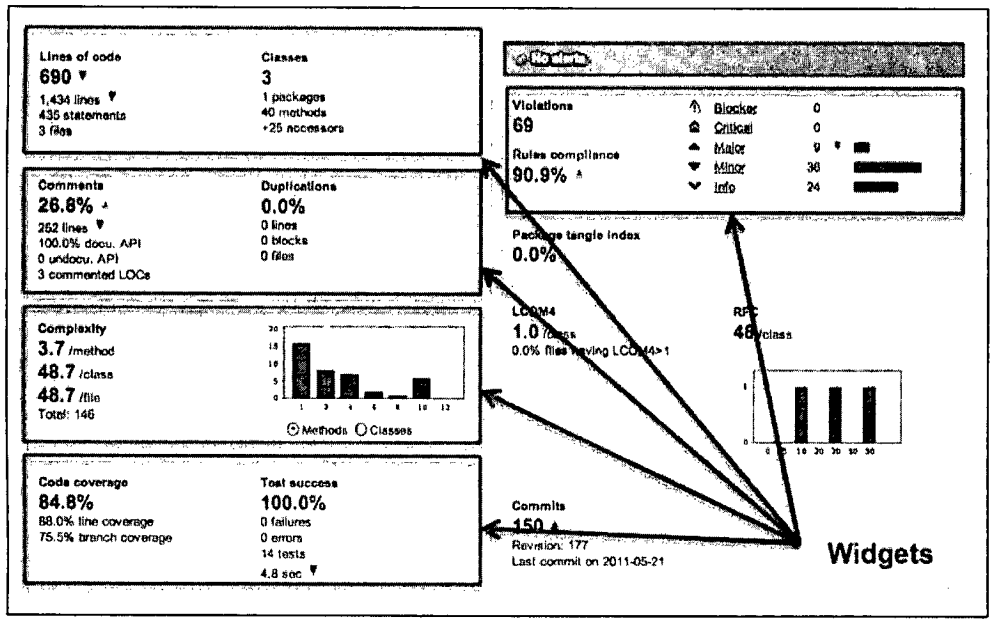
Todo esto se puede hacer a través del botón de "Gestionar filtros" (Manage filters) en la esquina superior derecha de la página de inicio.



5.1.6. CUADRO DE MANDOS DE UN PROYECTO

El cuadro de mandos de un proyecto ("Dashboard") es el punto de entrada cuando se revisa un proyecto. Presenta los resultados de las métricas del proyecto de forma general.

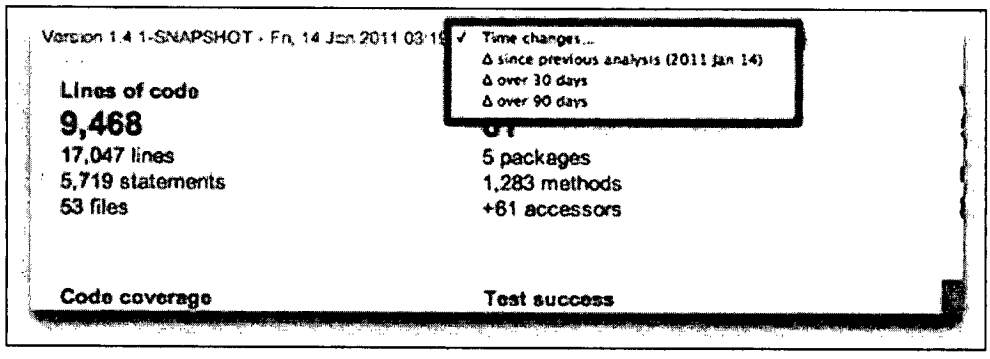
El cuadro de mandos se compone de elementos (cajas) que se denominan "widgets":



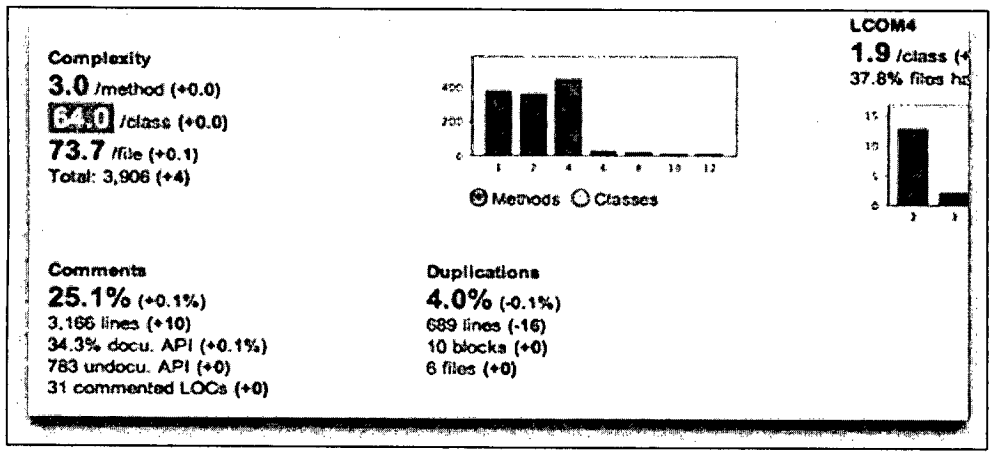
Un widget no es más que una representación agrupada de un conjunto de métricas que tienen una naturaleza similar.

Por ejemplo, existe un widget para ver las métricas asociadas al tamaño, un widget para ver las métricas asociadas a la documentación del código (comentarios) y el código duplicado, otro para ver métricas de pruebas unitarias, otro para cumplimiento de reglas, etc.

Como en los filtros, una de las funciones más útiles es la de vistas diferenciadas, que permite ver los cambios en las métricas de un proyecto en un periodo de tiempo. Basta con utilizar la lista desplegable "Time changes..." para mostrar el periodo deseado.



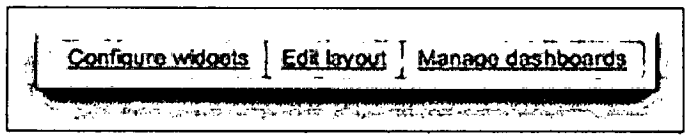
Cuando se selecciona un periodo de tiempo, entonces las métricas diferenciales aparecerán al lado de las métricas del proyecto:



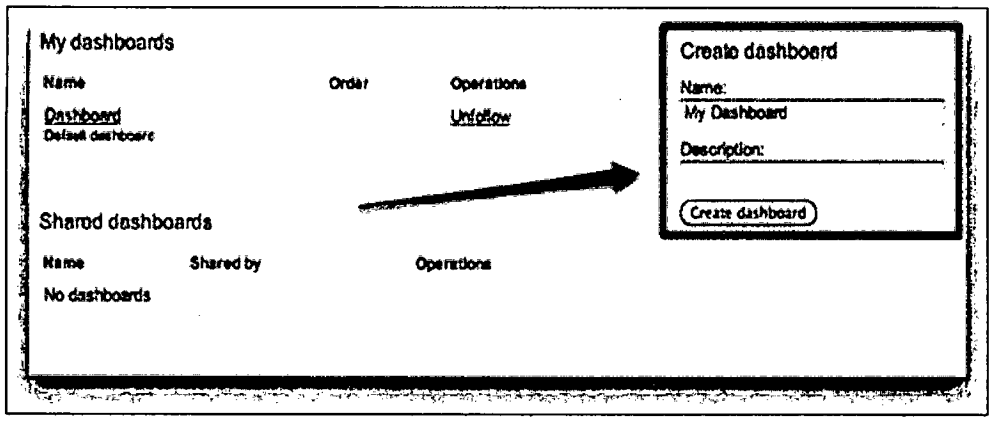
ADMINISTRAR LOS CUADROS DE MANDO

Al igual que los filtros, cualquiera puede crearse nuevos paneles de mando donde mostrar solo la información que más le interese.

La configuración se realiza a través de las opciones de la esquina superior derecha en el cuadro de mando de un proyecto.

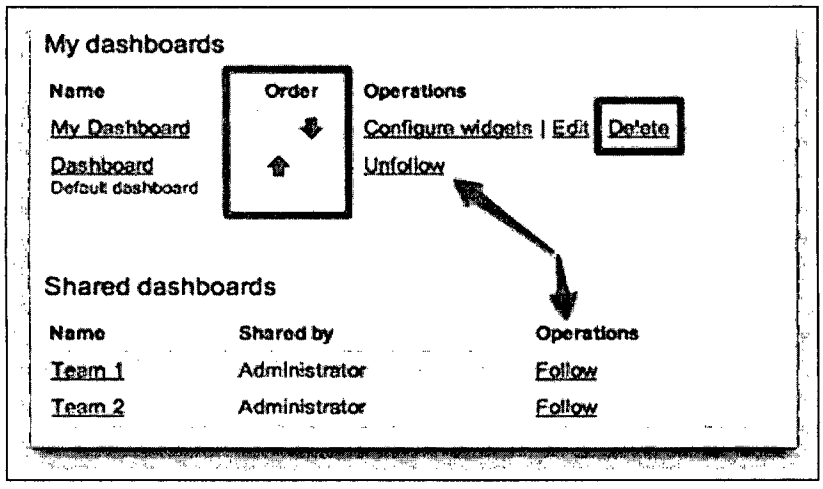


Selecciona "Gestionar cuadros de mando" (Manage Dashboards) y accederás al formulario para crear un nuevo cuadro de mando:



El nuevo cuadro de mando aparecerá inmediatamente en el menú lateral cuando estés dentro de un proyecto. Simplemente tendrás que añadir los diferentes widgets para componer el cuadro de mando.

Puedes administrar los cuadros de mando creados a través de la opción de "Manage dashboards":



Desde aquí puedes:

Cambiar el orden de los cuadros de mando que aparecerán en el menú lateral.

Borrar un cuadro de mando anteriormente creado.

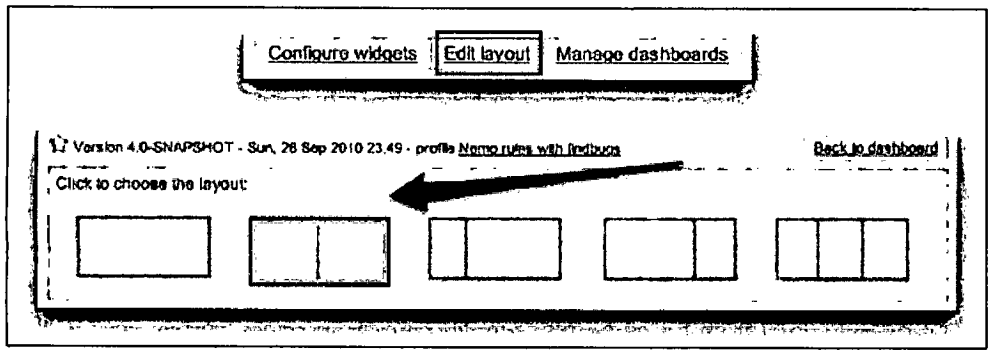
Seguir o dejar de seguir un cuadro de mando compartido por un administrador.

PERSONALIZAR UN CUADRO DE MANDO.

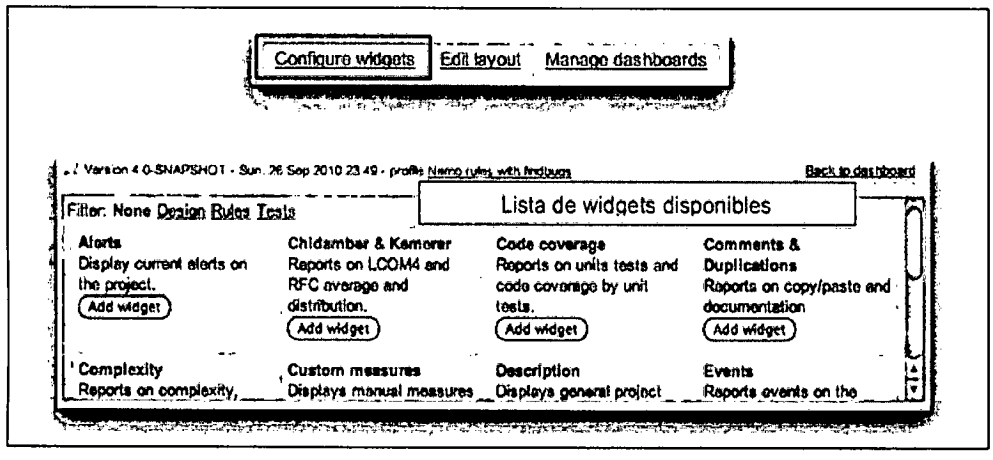
Cualquier cuadro de mando creado por el usuario se puede personalizar para incluir diferentes widgets e incluso para distribuirlos de la forma más conveniente.

El cuadro de mando por defecto solo puede ser modificado por un administrador.

Para cambiar la distribución de un cuadro de mando tienes que pinchar en "Edit layout" dentro de cualquier proyecto:

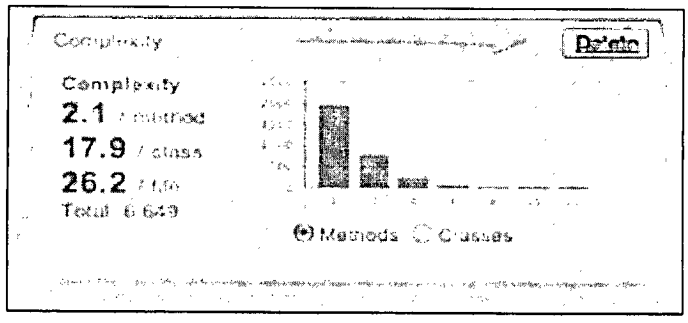


Para añadir widgets al cuadro de mando simplemente selecciona "Configure widgets". Se mostrará una lista de widgets disponibles en los que puedes ir pulsando "Add widget" para ir añadiendo los que desees.



Por último para mover un widget dentro de un cuadro de mando, desde la pantalla de configurar los widgets, basta con que pulses y arrastres el widget deseado a su nueva posición.

Para eliminar un widget tienes que seleccionar la pantalla de configuración de widgets y después pulsar "Delete":

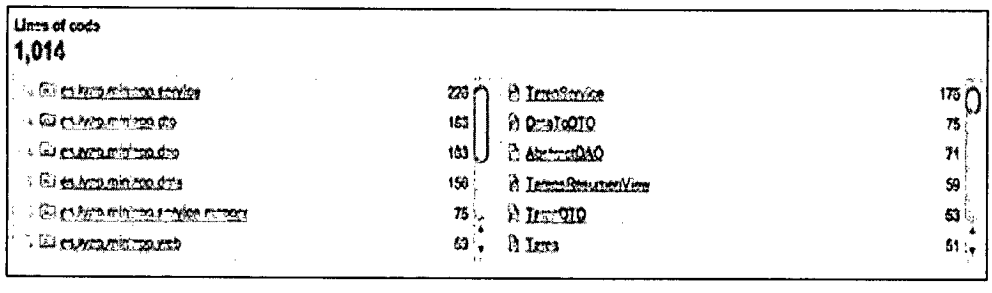


5.1.7. NAVEGACIÓN POR LAS MÉTRICAS

En cualquier proyecto, siempre es posible acceder a una navegación detallada en función de los valores obtenidos pinchando en la métrica concreta.

La herramienta mostrará la estructura por componentes, la jerarquía se representa de izquierda a derecha.

En la primera posición de la lista aparecerá el componente que tenga el mayor valor de la métrica que fue seleccionada (por ejemplo, la que más líneas de código tiene, la de mayor código duplicado, etc.)



Si pinchas en una clase te mostrará información detallada del recurso en concreto a través del visor de recursos ("Resource viewer").

5.1.8. EVENTOS

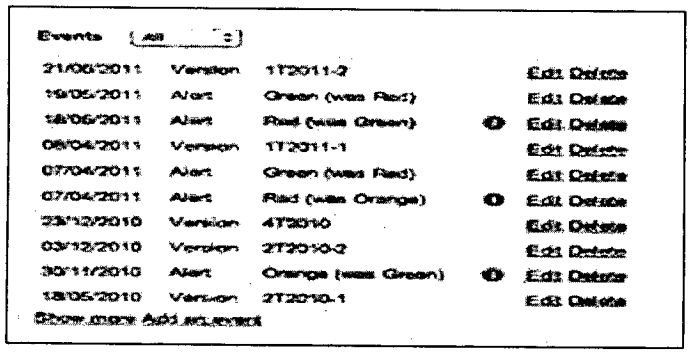
Por defecto puedes incluir en el cuadro de mandos de un proyecto un widget que refleja los eventos durante el ciclo de vida del proyecto.

Hay dos tipos de eventos:

Manuales: se puede añadir en el momento que se desea y con una categoría concreta para reflejar un hito importante en el proyecto.

Automatizados: se añaden mediante un mecanismo automático cuando hay un cambio de versión del proyecto o cuando se produce una alerta porque se ha superado un umbral.

La gestión de eventos se detalla en el apartado de Manual de Administración.



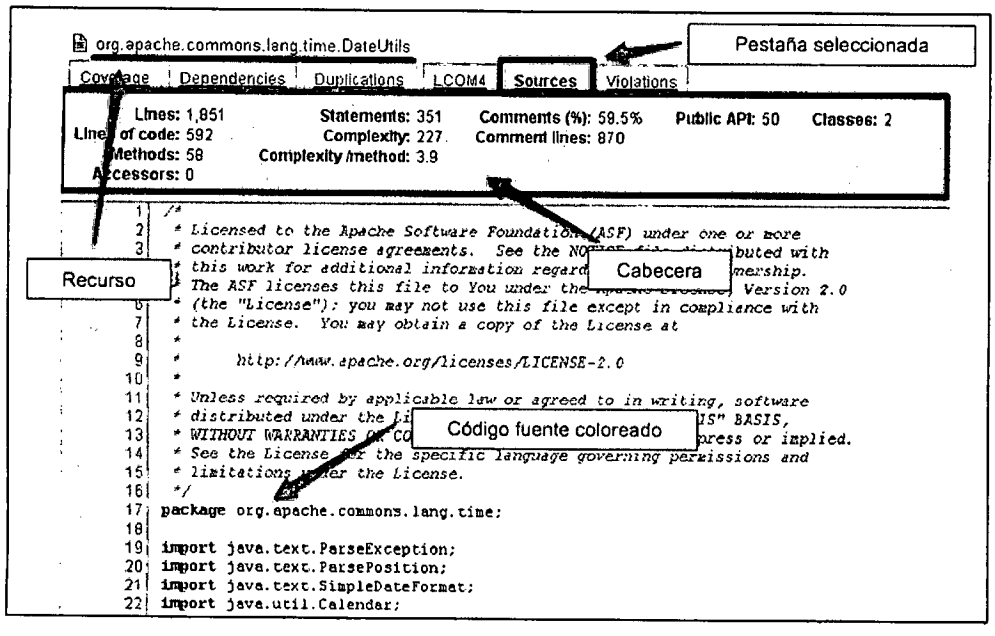
5.1.9. EL VISOR DE RECURSOS

El visor de recursos es el último elemento donde examinar la información más detallada de un recurso en concreto, como un fichero java o las pruebas unitarias. Lo más importante del visor es

que permite ver el recurso de acuerdo a diferentes aspectos de la calidad.

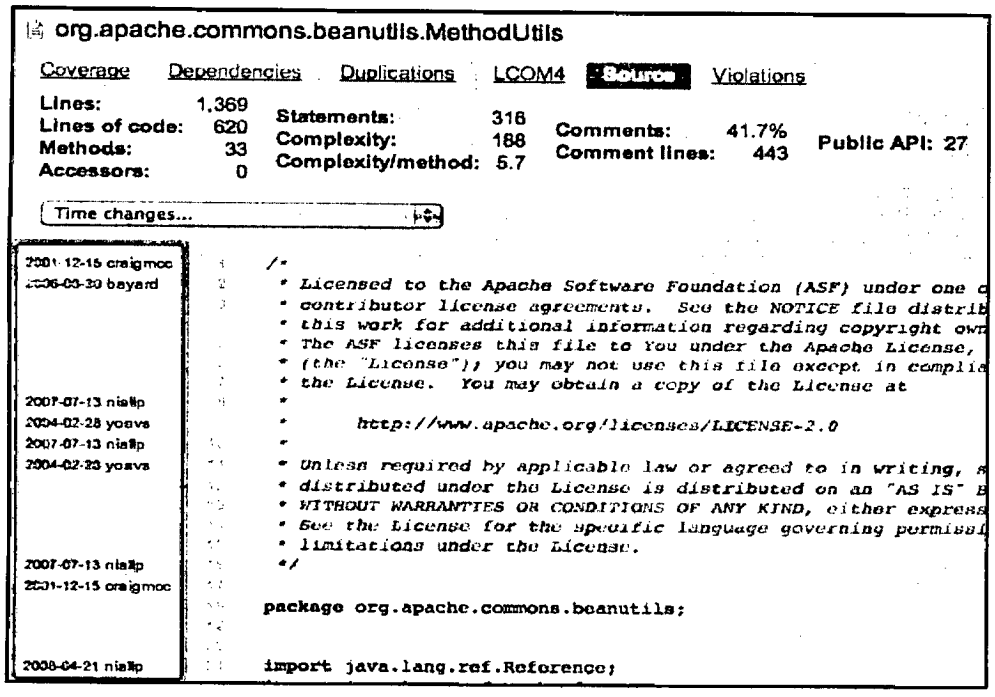
VISOR DE CÓDIGO FUENTE

Cuando navegas por las diferentes métricas llegará un momento en el que alcances el máximo nivel de detalle, en ese punto, y siempre que tengas los permisos adecuados serás capaz de visualizar el código fuente.



Las diferentes pestañas representan las diferentes vistas sobre el código fuente. Dependiendo de la pestaña seleccionada verás una cabecera con información específica. Y finalmente en el pie, el código fuente contextualizado según la pestaña seleccionada.

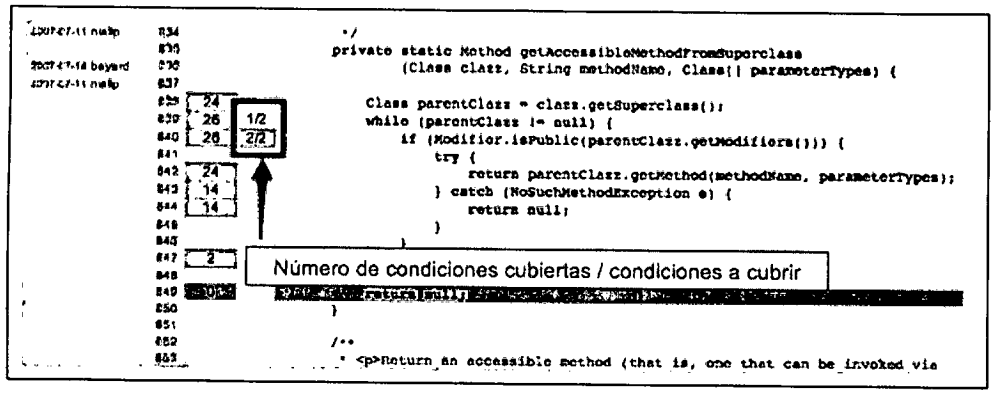
También es posible ver información del control de versiones en el visor de recursos gracias al SCM Activity Plugin.



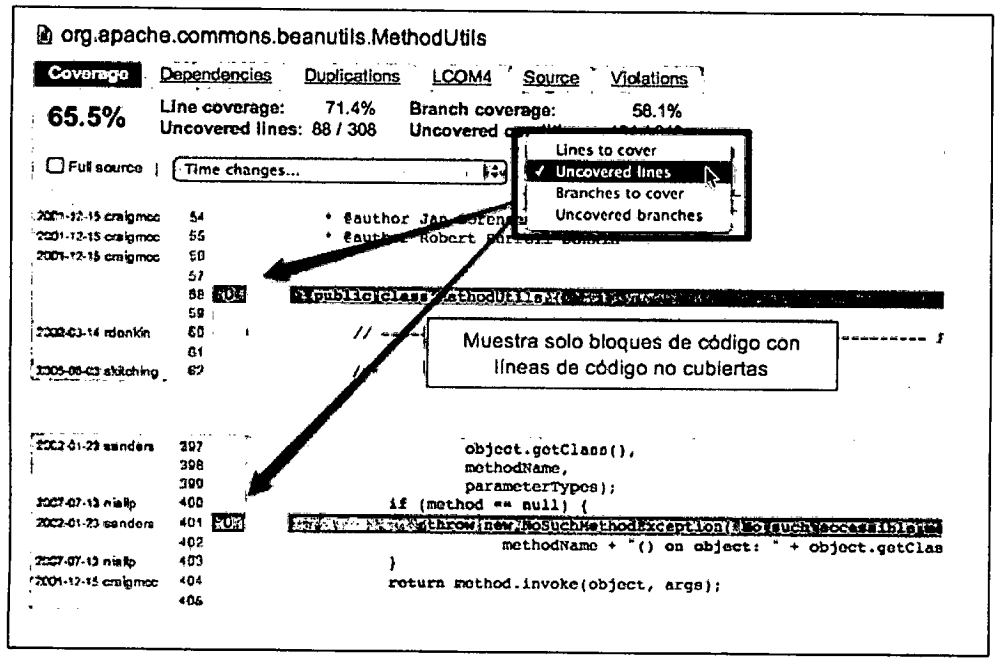
A continuación se describe el contenido de cada una de las pestañas:

La pestaña de cobertura (Coverage)

Esta pestaña muestra el código fuente incluyendo información extra y colores para visualizar la cobertura del código de las pruebas. En la primera columna aparece el número de veces que la ejecución de las pruebas ha pasado por ahí, y la segunda columna muestra el número de condiciones cubiertas del total de condiciones a cubrir.

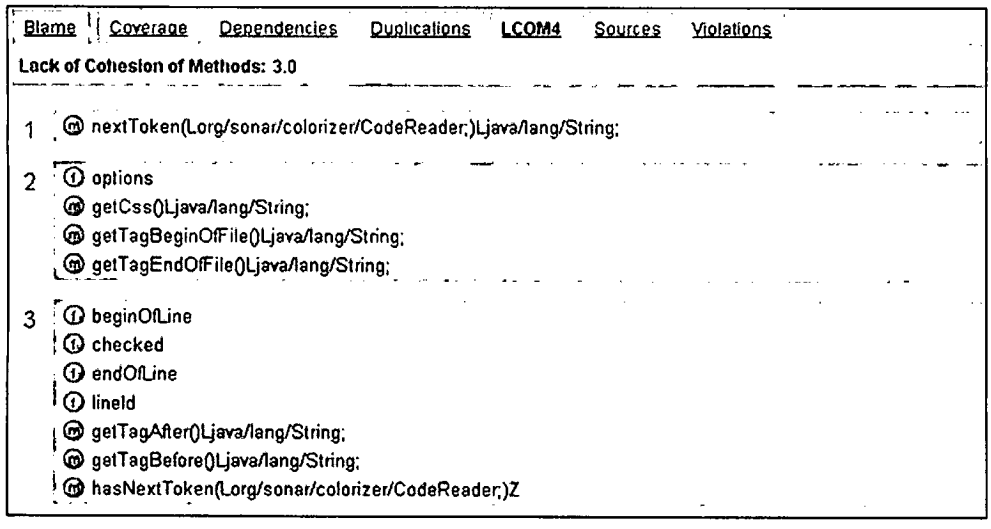


Una lista desplegable permite filtrar la información a mostrar:



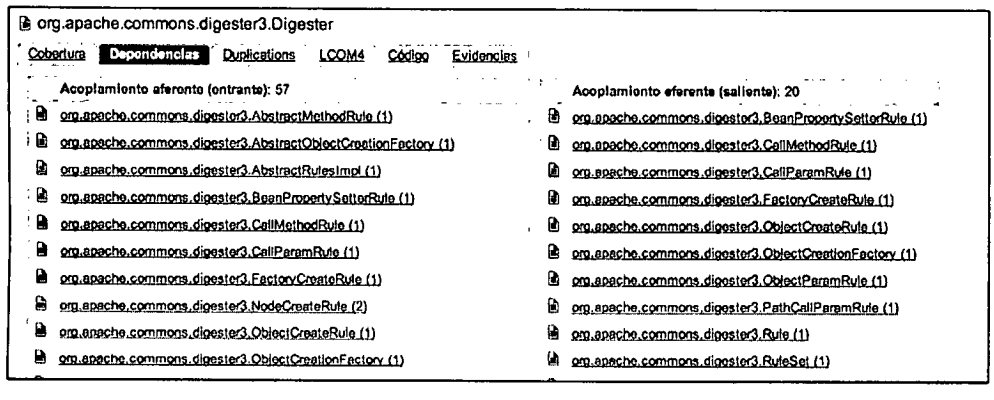
La pestaña de cohesión (LCOM4)

Muestra los diferentes componentes de un recurso y como están relacionados entre sí. Tener más de un bloque significa que el recurso está asumiendo más responsabilidad de la que debería, por lo que podría separarse en varios más independientes.



La pestaña de dependencias (Dependencies)

Muestra todas las dependencias de entrada y de salida en el recurso seleccionado:

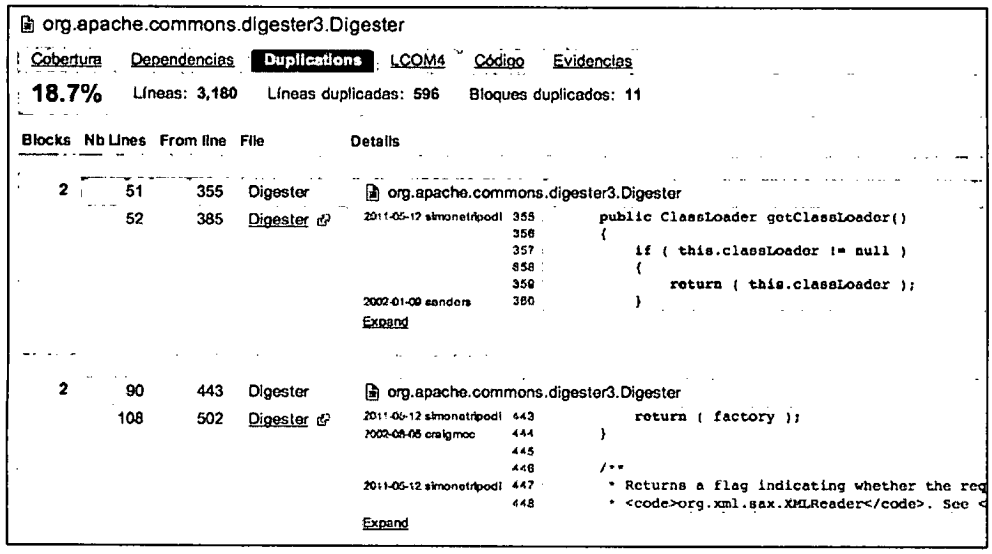


La información de "Afferent couplings" indica el número de clases que dependen de esta clase. Esto nos da información sobre la responsabilidad de la clase.

La información de "Efferent couplings" indica el número de clases de las que depende esta clase. Esto nos dice como de independiente es la clase en concreto.

La pestaña de duplicados (Duplications)

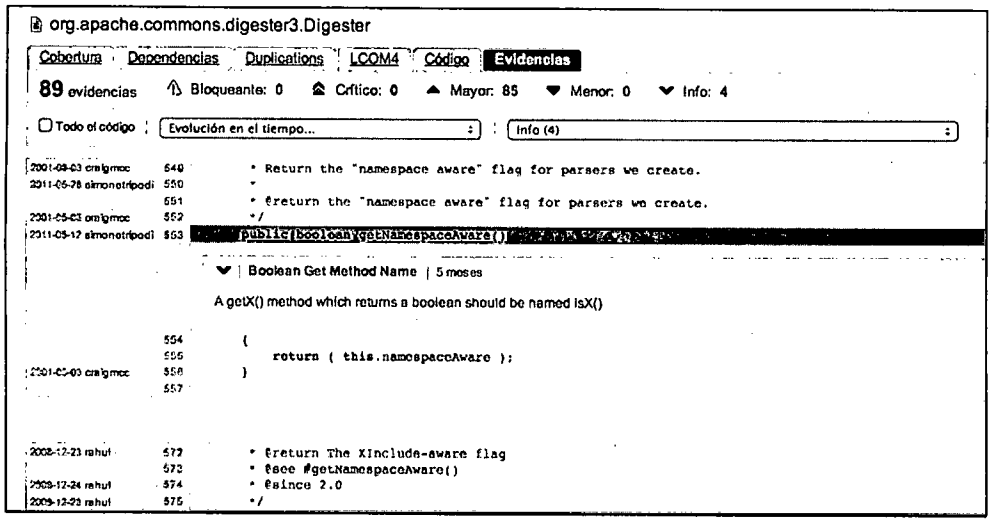
Muestra cada bloque de código duplicado dentro de una clase, así como el nombre de la otra clase con la que se está duplicando.



La pestaña de evidencias (Violations)

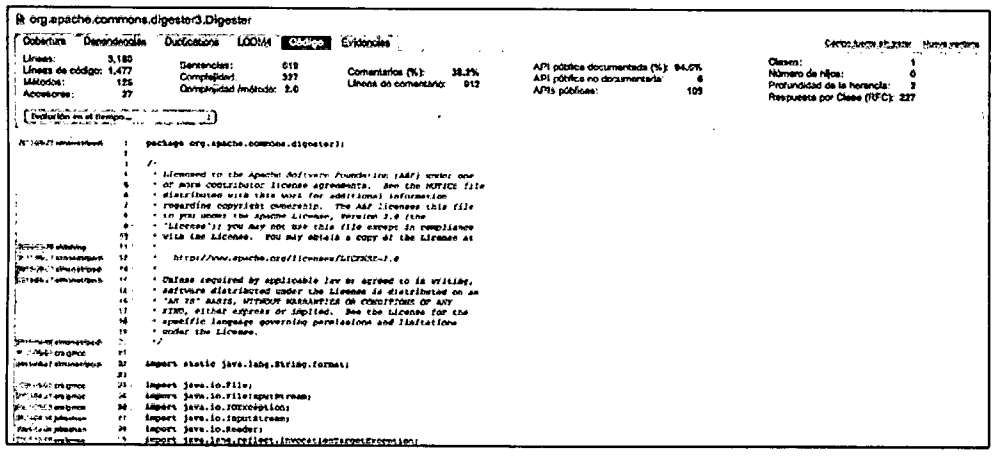
Muestra información anotada sobre el código fuente respecto al cumplimiento de reglas de buenas prácticas. Por defecto sólo se muestra el código cercano a donde se ha encontrado el incumplimiento.

Una lista desplegable permite filtrar por prioridades o por una evidencia específica.



La pestaña de fuentes (Sources)

Esta pestaña muestra el código fuente con la sintaxis coloreada para facilitar su lectura.



VISOR DE RESULTADOS DE LAS PRUEBAS UNITARIAS

Este visor muestra detalles de las métricas de las pruebas unitarias (excepto la cobertura) cuando alcanzas el mayor nivel de detalle para una clase que implementa una prueba unitaria.

org.apache.commons.digester3.DigesterTestCase

Código | Tests

100% Tests: 19 Fallos/Errores: 0/0 Duración: 4 ms

Duración	Nombre del test
0 ms	testNullFileParse
0 ms	testNullInputSourceParse
1 ms	testNullInputStreamParse
0 ms	testNullReaderParse
0 ms	testNullStringParse
0 ms	testNullURLParse
0 ms	testProperties

Existen dos pestañas diferentes para las clases de pruebas unitarias:

La pestaña de fuentes de las pruebas (Sources)

Esta pestaña muestra el código fuente de la clase que implementa la prueba tal cual.

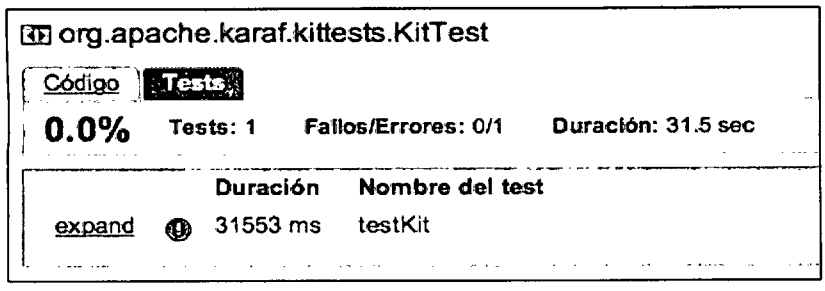
org.apache.commons.digester3.RuleTestCase

Código | Tests

```
1  /* $Id: RuleTestCase.java 1102402 2011-05-12 18:03:26Z simonetripodi $
2   *
3   * Licensed to the Apache Software Foundation (ASF) under one or more
4   * contributor license agreements.  See the NOTICE file distributed with
5   * this work for additional information regarding copyright ownership.
6   * The ASF licenses this file to You under the Apache License, Version 2.0
7   * (the "License"); you may not use this file except in compliance with
8   * the License.  You may obtain a copy of the License at
9   *
10  *      http://www.apache.org/licenses/LICENSE-2.0
11  *
12  * Unless required by applicable law or agreed to in writing, software
13  * distributed under the License is distributed on an "AS IS" BASIS,
14  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
15  * See the License for the specific language governing permissions and
16  * limitations under the License.
17  */
18
19 package org.apache.commons.digester3;
20
21 import static org.junit.Assert.assertEquals;
22 import static org.junit.Assert.assertNotNull;
23 import static org.junit.Assert.assertNull;
24 import static org.junit.Assert.assertTrue;
25 import static org.junit.Assert.fail;
26
27 import java.io.IOException;
28 import java.io.InputStream;
29 import java.util.ArrayList;
30
31 import org.apache.commons.digester3.Digester;
32 import org.apache.commons.digester3.ExtendedBaseRules;
33 import org.apache.commons.digester3.RuleSet;
34 import org.junit.After;
35 import org.junit.Before;
36 import org.junit.Test;
37 import org.xml.sax.SAXException;
```

La pestaña de las pruebas (Tests)

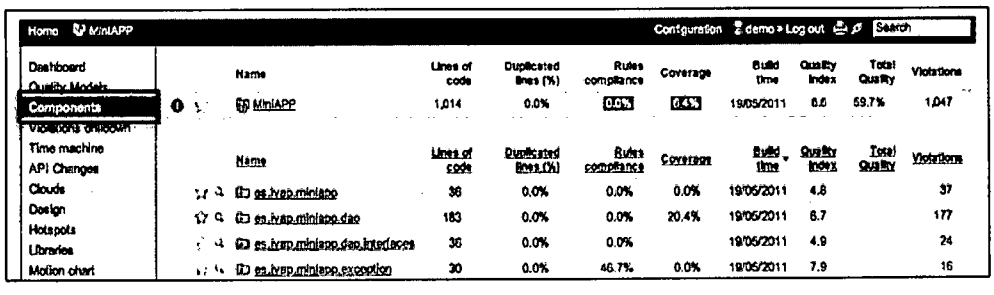
Esta pestaña muestra los resultados de las pruebas unitarias que se implementan en esta clase. En el caso de que la prueba falle o de error se puede mostrar cual ha sido la causa del error.



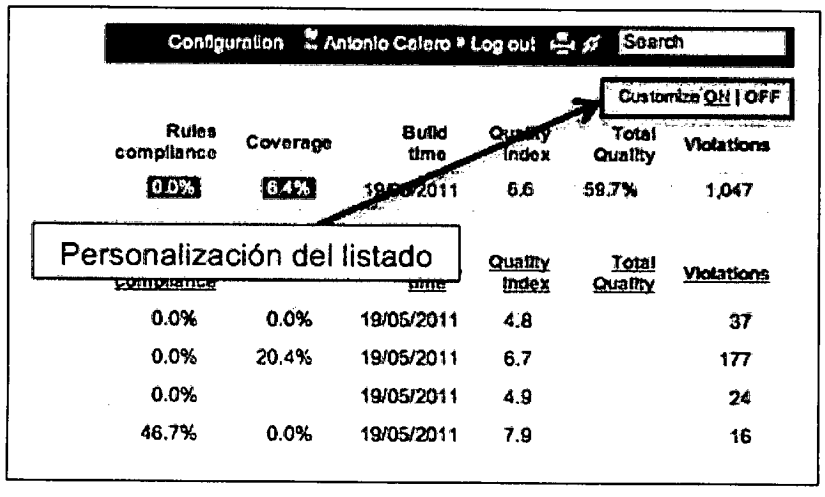
5.1.10. COMPONENTES

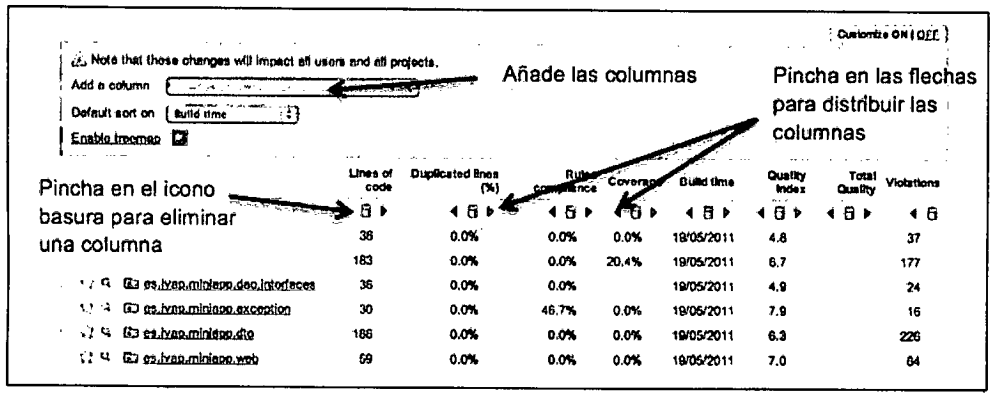
Desde cualquier punto de la navegación por un proyecto puedes acceder a un listado de sus componentes a través del menú lateral.

Esta opción te lleva a una página similar a la página de inicio, sin embargo, en lugar de los filtros se muestran los componentes del proyecto.



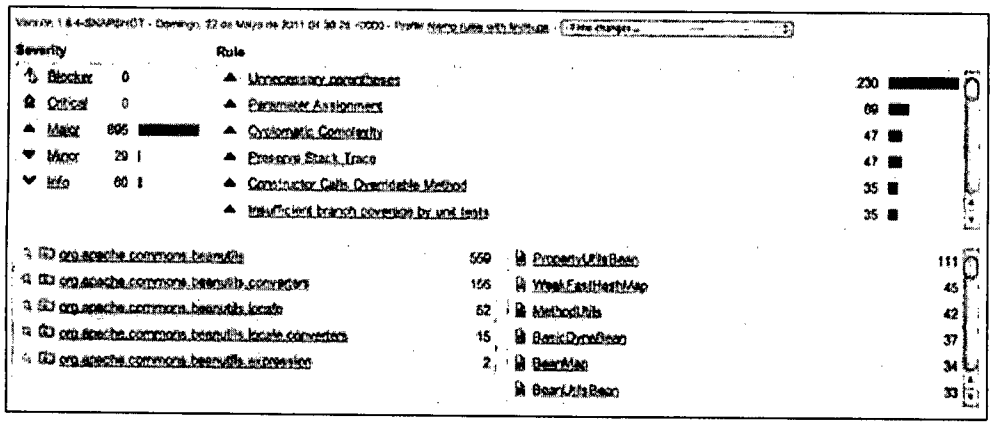
Al igual que en la página de inicio el listado muestra un conjunto de métricas que son totalmente configurables siempre que el usuario sea administrador:





5.1.11. NAVEGACIÓN POR LAS EVIDENCIAS

Se puede acceder a la navegación por las diferentes evidencias de incumplimiento de buenas prácticas (en base a las reglas de análisis estático) a través de la opción correspondiente del menú (Violations drilldown).

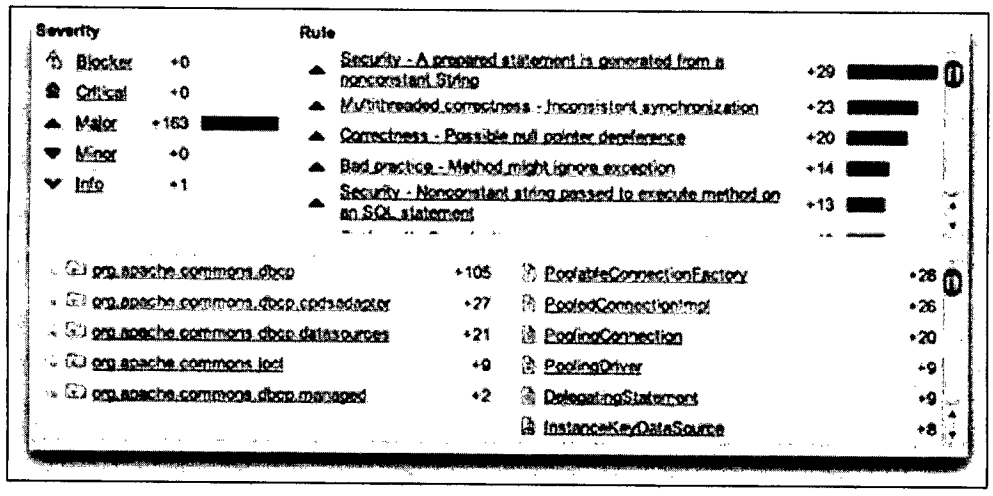


La primera parte permite seleccionar un criterio para mostrar las evidencias: por prioridades o con seleccionando una regla específica.

La siguiente sección representa la jerarquía de recursos del proyecto de izquierda a derecha. Al principio de cada lista aparecerá el componente y recurso con mayor número de evidencias según el criterio seleccionado.

Pinchando en una clase concreta accederás al visor de recursos donde consultar la información del elemento con el mayor nivel de detalle.

Una de las opciones más interesantes a la hora de navegar por las evidencias es la de ver las diferencias con respecto a análisis anteriores o en un periodo específico de tiempo. Puedes consultar las nuevas evidencias encontradas pero no las eliminadas. Para ello puedes seleccionar el periodo que te interese desde la lista desplegable "Time changes".



Para determinar cuando aparece una nueva evidencia se ejecuta el siguiente algoritmo en cada inspección del código:

Para cada evidencia, si existía una evidencia de incumplimiento de la misma regla y estaba en el mismo número de línea y con el mismo identificador -> no es nueva.

Para cada evidencia, si existía una evidencia de incumplimiento de la misma regla y tiene el mismo identificador -> no es nueva.

Para cada evidencia, si existía una evidencia de incumplimiento de la misma regla y estaba en el mismo número de línea pero con identificador diferente -> no es nueva.

En cualquier otro caso, se tratará como una nueva evidencia.

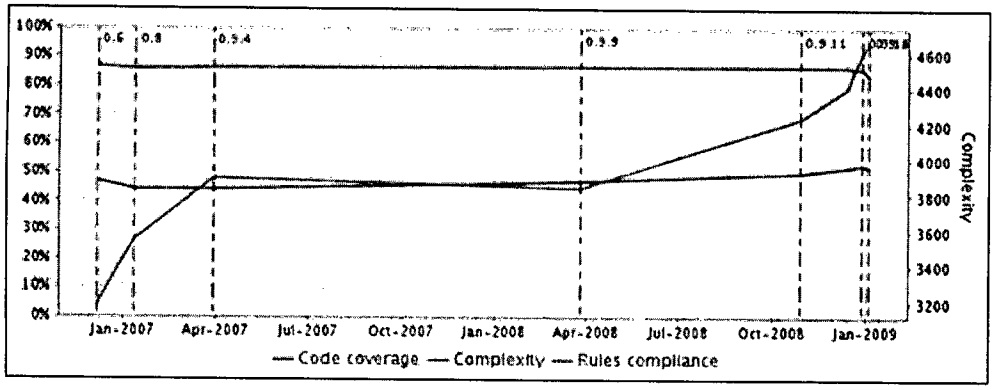
5.1.12. LA MÁQUINA DEL TIEMPO

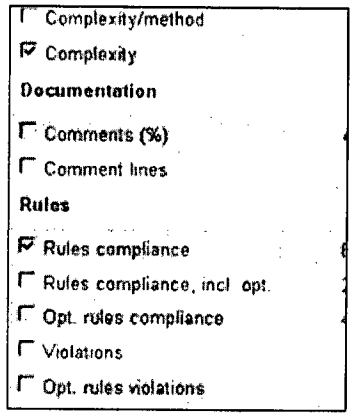
Desde cualquier punto de la navegación por un proyecto puedes acceder a la funcionalidad de la máquina del tiempo (Time machine) para estudiar la evolución de un proyecto.

La máquina del tiempo permite revivir el pasado analizando todos los análisis realizados. La página se divide en cuatro secciones.

GRÁFICA PERSONALIZADA

La gráfica representa la evolución desde el primer análisis realizado hasta el último análisis. En la gráfica aparecerán todas las métricas que se encuentren seleccionadas de todo el conjunto de métricas disponible.







Para cambiar las métricas que se representan en la gráfica basta con seleccionar las métricas que te interesen y pulsar el botón de "comparar". La gráfica se regenerará con las métricas elegidas.

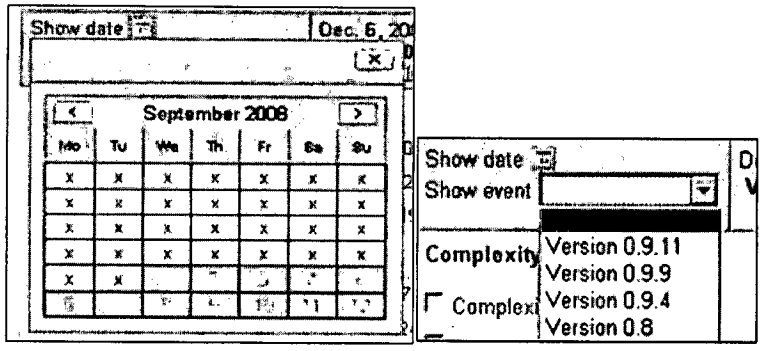
Si quieres que las métricas seleccionadas sean las de por defecto, el usuario administrador dispondrá de una opción adicional "Set as default" que podrá utilizar.

SELECCIÓN DE RESULTADOS

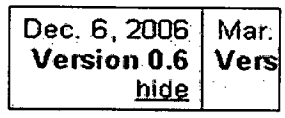
Cuando se utiliza la máquina del tiempo, siempre se muestra la primera "foto" del proyecto (el primer análisis) y despues los últimos cinco análisis.



Puedes añadir fotos a esa vista, o bien a través de la fecha, o bien a través de un evento (una nueva versión, la superación de un umbral, etc.)



También es posible ocultar una foto de la vista utilizando el enlace "hide" al lado de cada una de las fotos que se muestran en la máquina del tiempo:



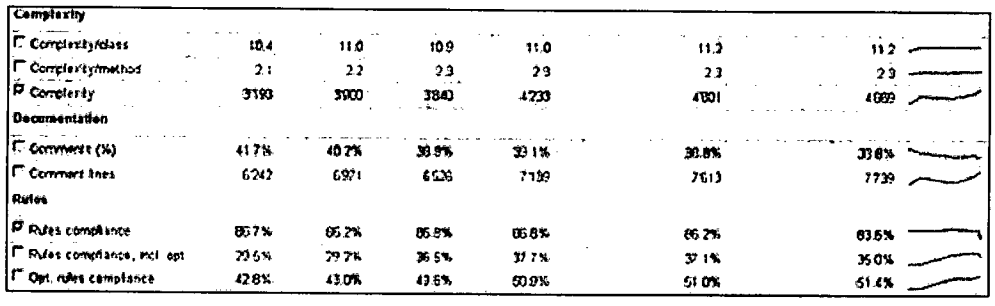
Es importante destacar que tanto como si ocultas columnas como si añades nuevas, eso no tendrá ninguna influencia sobre la gráfica personalizada.

LA TABLA DE MÉTRICAS

Esta tabla muestra la lista de todas las métricas disponibles en el sistema, agrupadas por su naturaleza, y también incluyendo las métricas manuales.

Cada columna representa una foto que mostrará el valor medido para esa métrica.

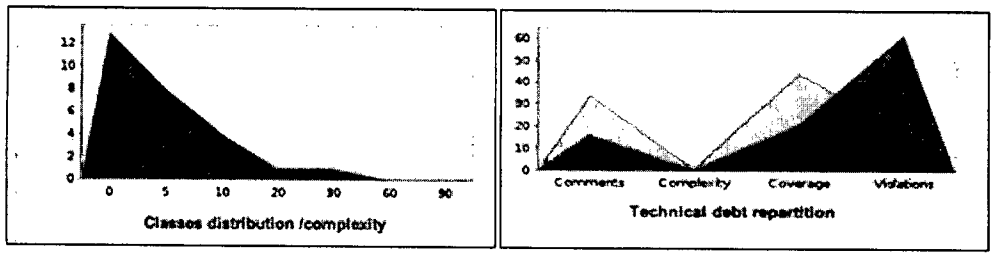
A la derecha de la fila se puede observar una representación de la tendencia que sigue la métrica correspondiente.



LA GRÁFICA DE DISTRIBUCIONES

Algunas de las métricas disponibles se representan a través de distribuciones, por ejemplo, la distribución de la complejidad por clases, la distribución de la cohesión entre las clases, la distribución de la deuda técnica, ...

Estas gráficas se muestran al final de la página de la máquina del tiempo:



5.1.13. NUBES

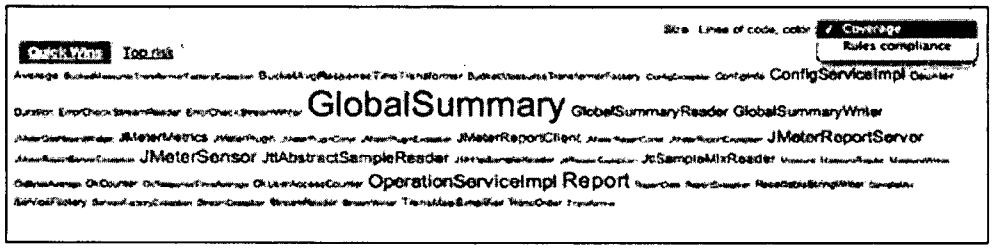
En cualquier proyecto puedes obtener una vista en forma de nubes de etiquetas para detectar puntos clave. Para ello basta con pulsar en la opción del menú lateral "Clouds" (Nubes).

Una vez dentro de la página con la nube de recursos puedes elegir dos pestañas:

Top risk: en este caso el tamaño representa la complejidad media por método en la clase, y el color representa la cobertura de código o el nivel de cumplimiento de reglas.

Quick wins: el tamaño representa las líneas de código de la clase. El color representa igualmente la cobertura o el nivel de cumplimiento.

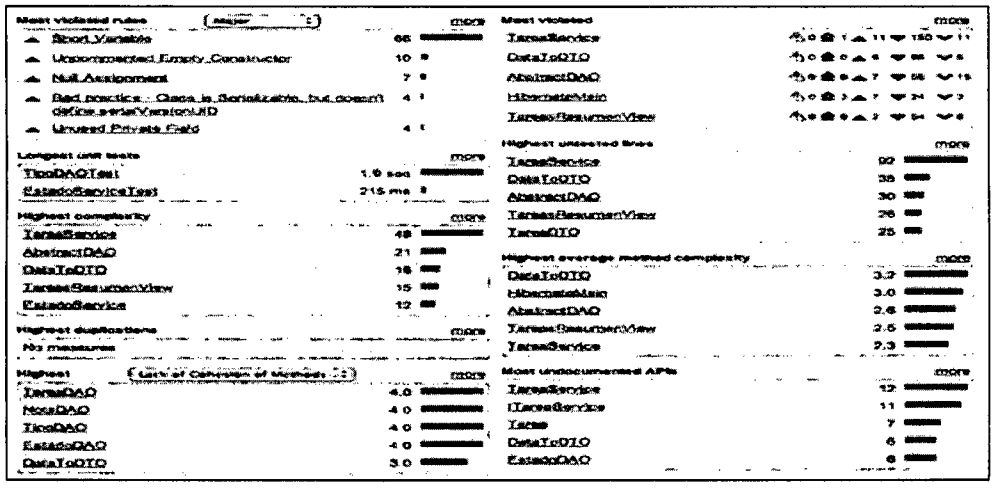
Si pinchas en una clase se abrirá una nueva ventana con el visor de recursos para analizar los detalles de la clase.



5.1.14. LOS "TOP"

En la opción de menú "Hotspots" (Puntos críticos) se puede ver el listado de las principales métricas con el ranking de los cinco primeros recursos afectados.

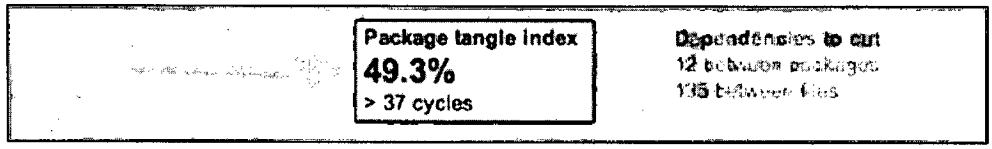
Es una manera muy rápida de obtener una foto de por donde empezar a revisar un proyecto.



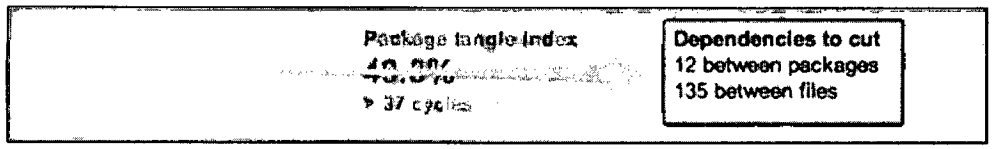
5.1.15. DISEÑO Y DEPENDENCIAS

Navegando por la herramienta encontrarás múltiples puntos en los dispondrás de información sobre el diseño y la arquitectura del sistema analizado. Uno de los aspectos más importantes es evitar ciclos entre clases y paquetes, ya que esto impide tener componentes mucho más independientes, que podrían ser cambiados.

En el cuadro de mandos podrás ver un widget que informa precisamente de esa cuestión:



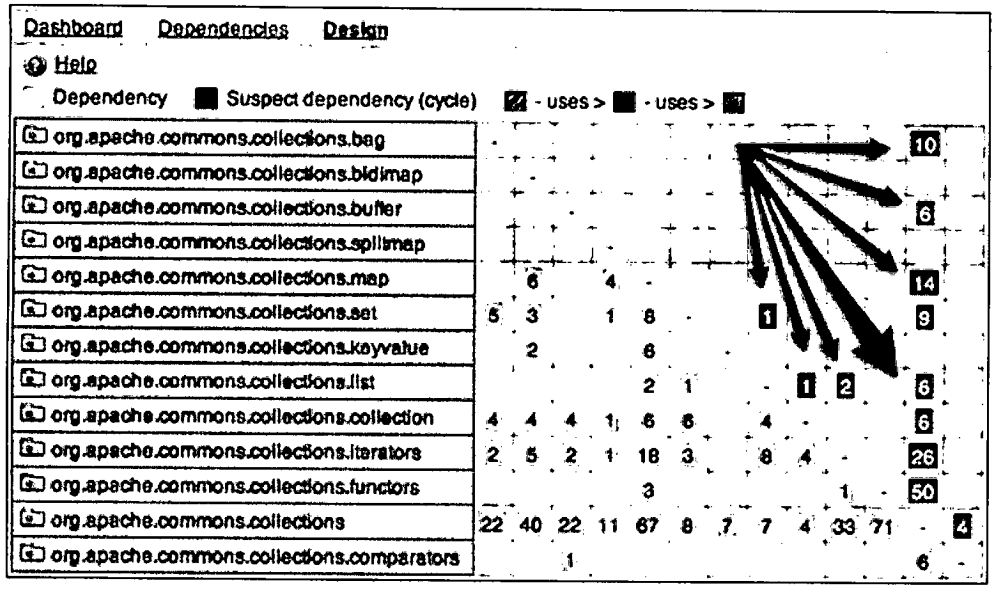
Esto nos da información exacta de cuantos ciclos existen en el proyecto. Cuanto mayor sea este valor peor es la arquitectura del sistema.



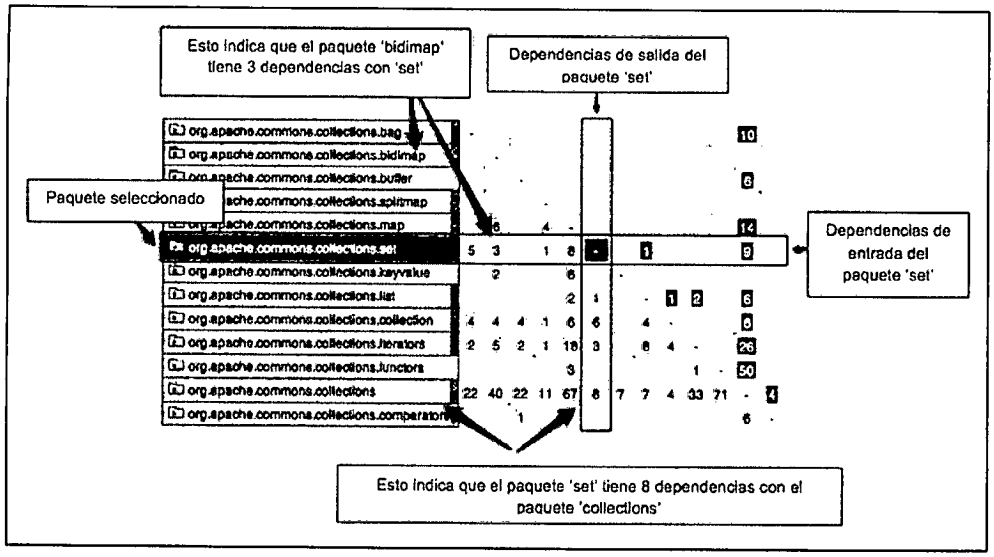
Podemos saber exactamente como resolver este problema si observamos el número de dependencias a cortar para eliminar esos ciclos.

Si hay muchos ficheros involucrados significa que el esfuerzo para eliminar las dependencias será mucho mayor.

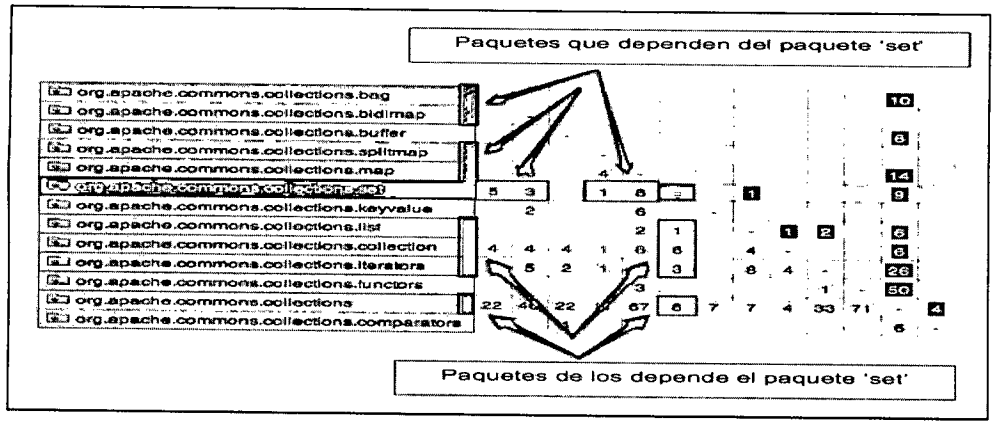
Para saber por donde empezar puedes pinchar en la métrica o acceder a la opción en el menú lateral de "Design" y acceder a la matriz de dependencias del proyecto donde se puede ver en rojo las dependencias cíclicas.



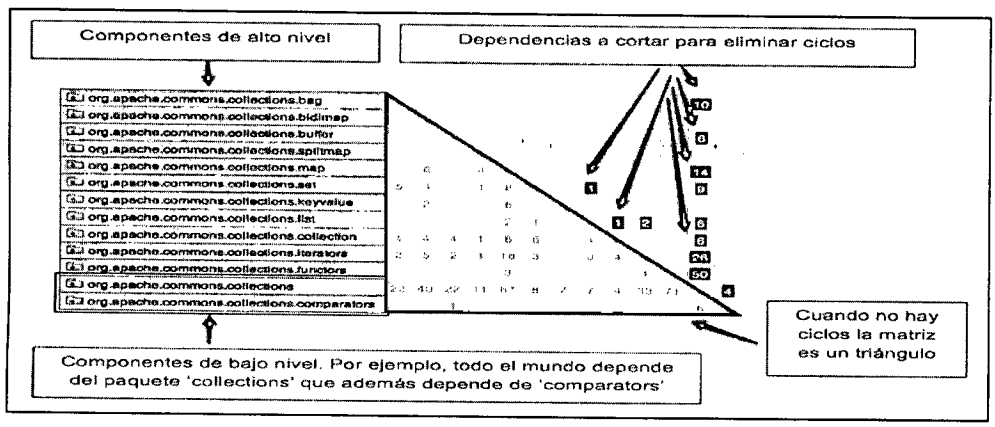
Puedes seleccionar cualquier componente y ver sus dependencias pinchando en su nombre:



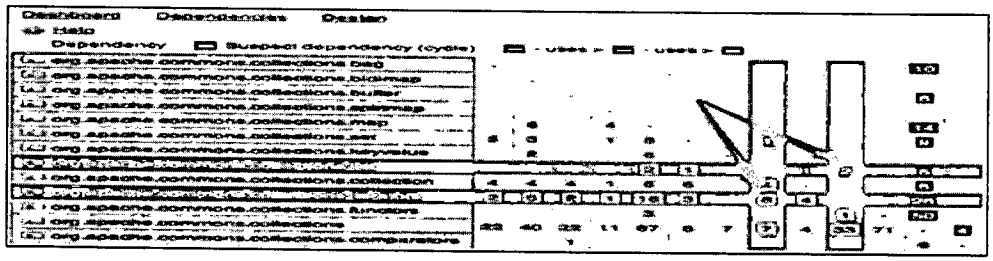
Los colores ayudan a ver las dependencias de entrada y de salida:



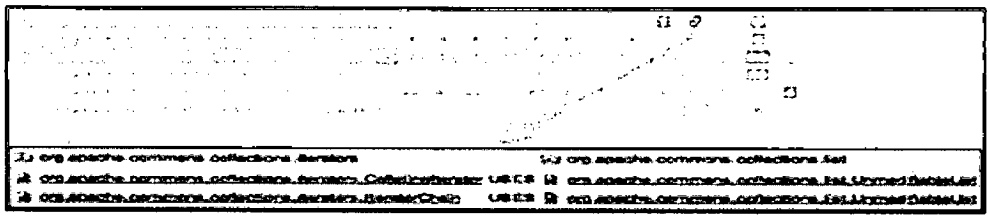
Con un vistazo rápido podemos saber cuales son los paquetes más independientes, los que más se utilizan y si existen ciclos entre ellos.



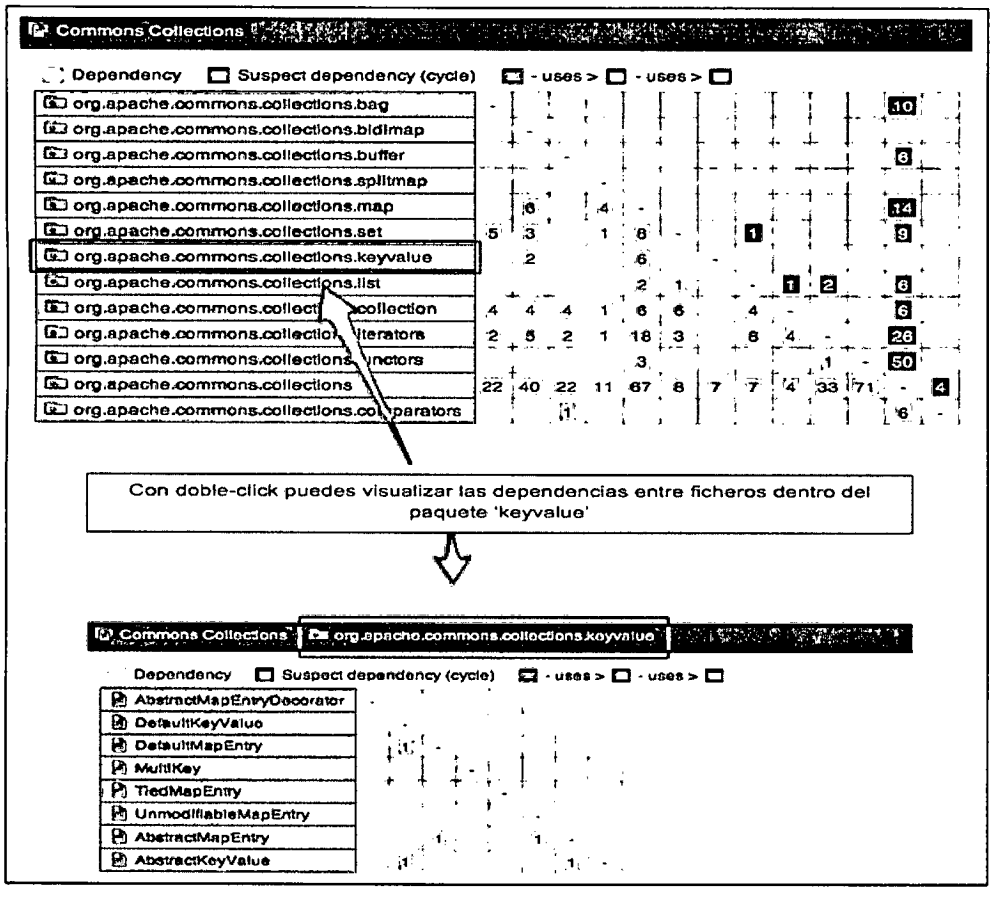
Para obtener más información sobre las dependencias basta con seleccionar el ciclo sospechoso:



Y haciendo doble click podemos llegar a saber incluso cuales son los ficheros o clases involucrados, de manera que podemos comenzar con la refactorización para corregirlo:



También se puede hacer un zoom específico en cada componente para obtener las dependencias entre sub-componentes:



5.1.16. LIBRERÍAS

Desde el punto de vista técnico de un proyecto, raramente el software se compone únicamente de un conjunto de ficheros con el código fuente. La reutilización de código ya hecho por cualquier otro equipo de desarrollo puede beneficiar mucho el proceso de desarrollo. Por eso si se trata de evaluar la calidad de un proyecto las dependencias con librerías externas son muy importantes.

Por ejemplo, se podría utilizar por las siguientes razones:

- Una determinada versión de una librería no se puede utilizar porque se sabe que contiene errores críticos.
- Se necesita conocer que dependencias transitivas explican la utilización de una librería concreta.
- Una librería de uso comercial no debe ser utilizada porque no se disponen de licencias.

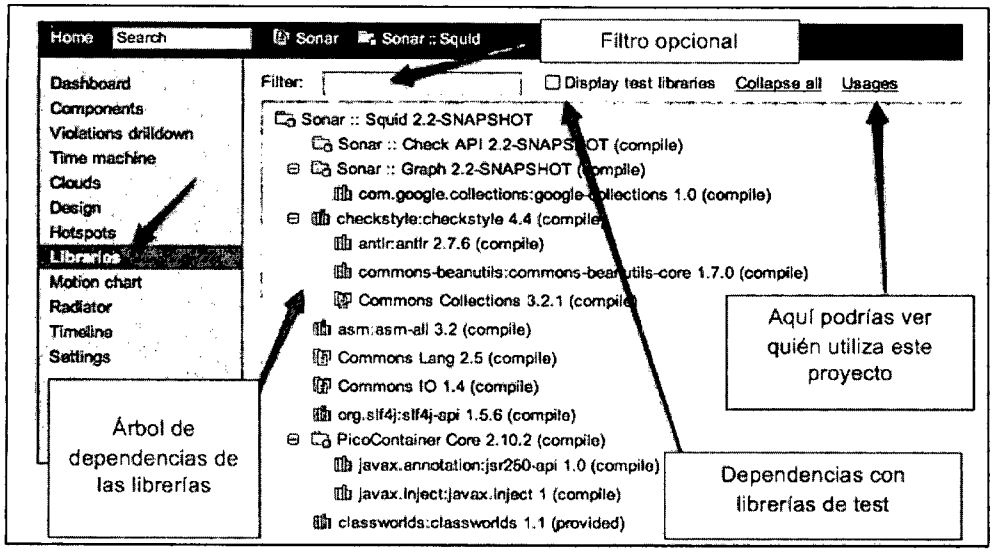
La opción para poder analizar estas situaciones solo está disponible cuando se trata de proyectos Java construidos con Maven.

Los detalles son:

LIBRERÍAS DE UN PROYECTO

Una vez seleccionando un proyecto, puedes acceder a las librerías que utiliza pinchando en el menú lateral, en la opción "Libraries".

Esta opción te permite ver rápidamente el árbol de dependencias del proyecto, incluyendo un filtro para buscar más rápidamente.

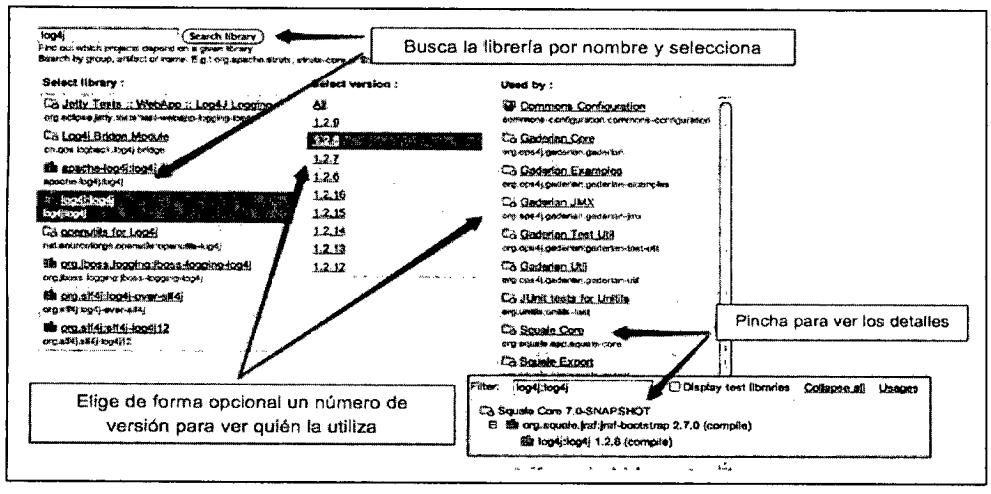


	El código fuente de esta librería no ha sido analizado
	El código fuente ha sido analizado y el proyecto depende de la última foto analizada
	El código fuente ha sido analizado pero el proyecto depende de una versión antigua

DEPENDENCIAS ENTRE PROYECTOS Y LIBRERÍAS

Conocer las librerías de un proyecto es muy útil pero todavía más interesante es poder revisar todos los proyectos y conocer cuales dependen de una librería con una versión concreta.

Esto puede llevarse a cabo a través de la opción "Dependencies" de la página de inicio:



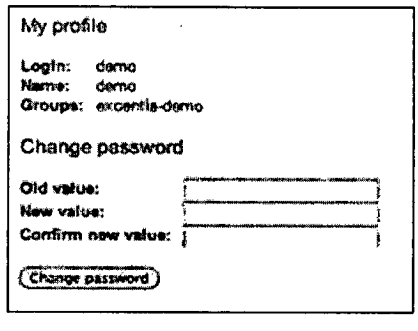
5.1.17. SEGURIDAD

La seguridad implementada en la herramienta es bastante sencilla y estándar. Es posible crear tantos usuarios y grupos como se desee, y los usuarios pueden pertenecer a un grupo o a varios.

Un usuario o un grupo puede tener múltiples roles, que dan acceso a varios servicios y funcionalidad dentro de la herramienta.

CAMBIO DE CONTRASEÑA

Puedes cambiar la contraseña pinchando en tu nombre de usuario en la esquina superior derecha. Tendrás que introducir la antigua contraseña, la nueva y la confirmación de la nueva.



ROLES

Existen cuatro roles en la herramienta. Uno es global y los otros tres se asocian a un proyecto:

- Administradores globales (Global administrators): tendrán la capacidad de realizar todas las operaciones de administración en la instancia, desde la configuración global, hasta la personalización de cuadros de mando, filtros por defecto, etc.
- Administrador de proyecto (Project administrators): tendrán la capacidad de realizar todas las operaciones sobre la configuración de un proyecto específico.
- Usuario de proyecto (Project user): tendrán la capacidad de navegar por todos los servicios de un proyecto, excepto ver el código fuente y la configuración.
- Revisor de código de proyecto (Project Code Viewers): tendrán la capacidad de ver el código fuente de un proyecto.

GRUPO "ANYONE"

Se trata de un grupo especial que siempre existe en cualquier instancia y que no puede modificarse.

Todos los usuarios pertenecen a ese grupo.

5.1.18. REVISIONES MANUALES DE CÓDIGO

El objetivo es ser capaces de asignar una evidencia a un usuario y comenzar una discusión que debería llegar a resolver el problema detectado.

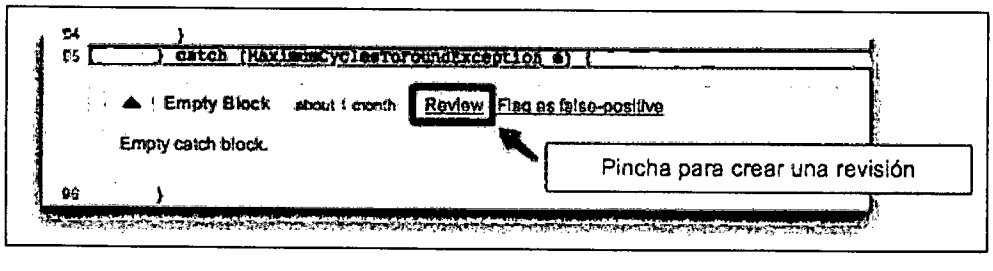
Una revisión puede tener dos estados: Abierta ("Open") o Cerrada ("Closed").

La herramienta automáticamente se encarga de cerrar todas aquellas revisiones tan pronto como su evidencia haya sido resuelta.

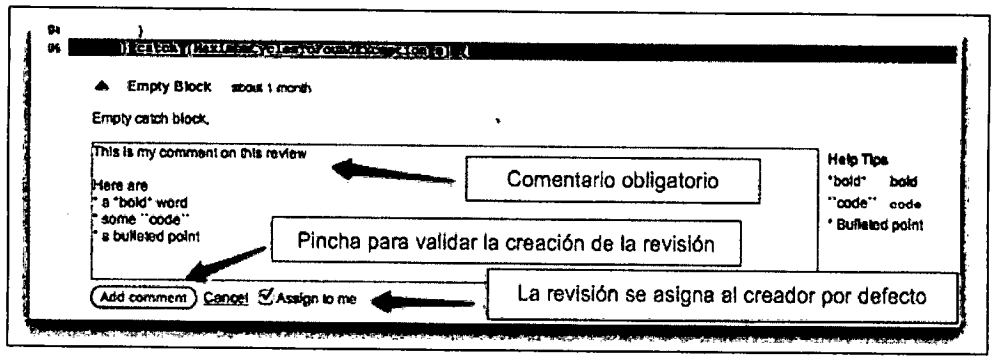
CREAR UNA REVISIÓN

Para crear una revisión tienes que estar identificado en la herramienta y debes de tener el rol de usuario (grupo "Users") dentro del proyecto donde se encuentra la evidencia que deseas revisar.

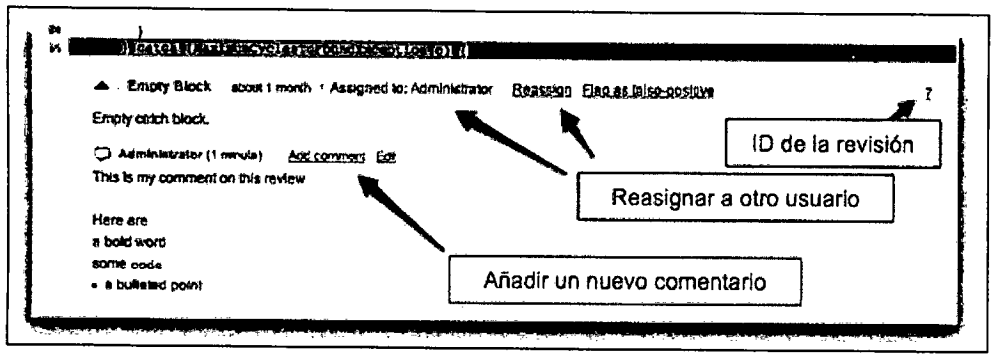
Directamente en el visor de recursos, cuando pases el ratón por encima de la evidencia aparecerá un nuevo enlace para comenzar a crear una nueva revisión.



Tendrás que añadir un comentario antes de crear realmente una revisión, pinchando en el botón "Add comment":



Por defecto, una revisión se asigna a su creador. Si no se selecciona, la revisión se crea pero no se asigna a ningún usuario. Una vez que se ha creado, cualquier usuario con permisos en el proyecto podrá ver la revisión debajo de la evidencia:

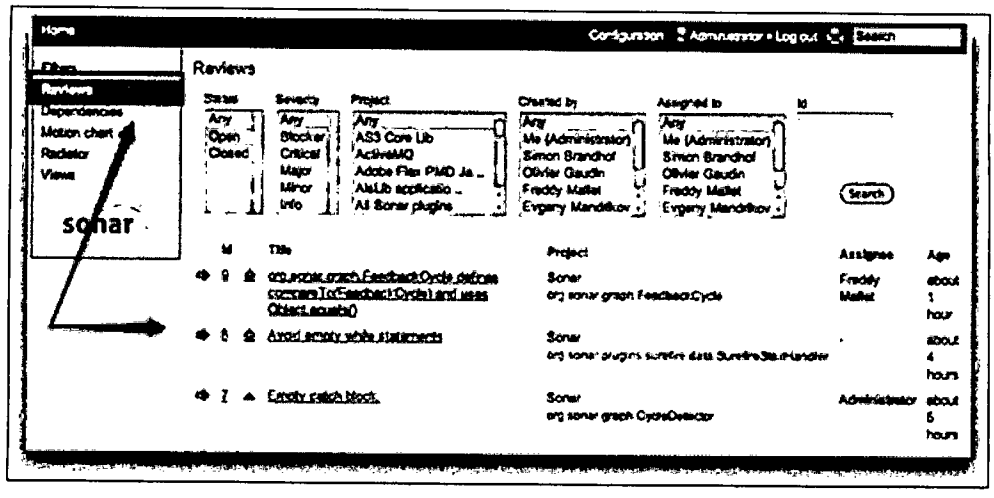


Solamente se puede editar el último comentario de una revisión por su propio creador.

BUSCAR REVISIONES

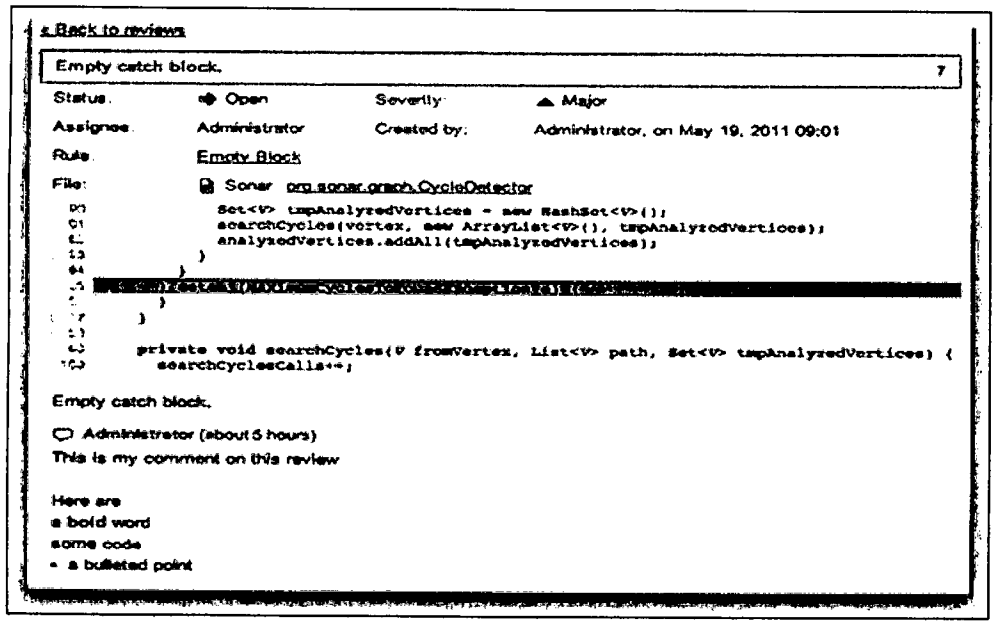
La página de revisiones accesible desde la página de inicio te permite buscar cualquier revisión indicando parámetros

opcionales específicos como puede ser el proyecto, el autor de la revisión, la prioridad de las evidencias, ...



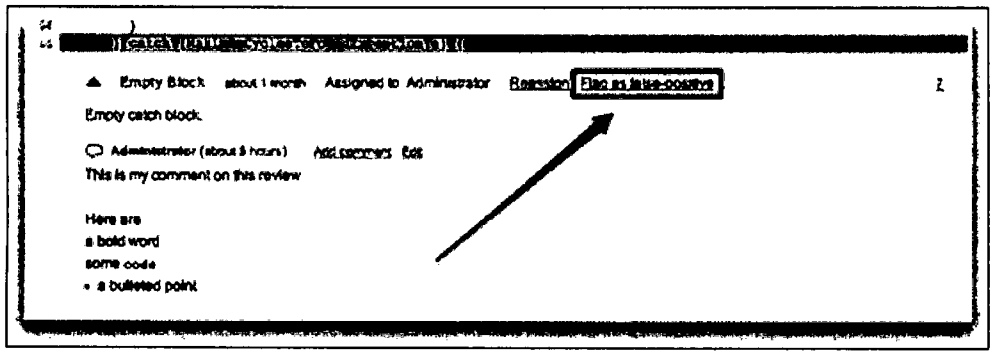
Por defecto se muestran todas las revisiones asignadas al usuario actual.

Pinchando en el titulo de la revisión puedes acceder a todos sus detalles:

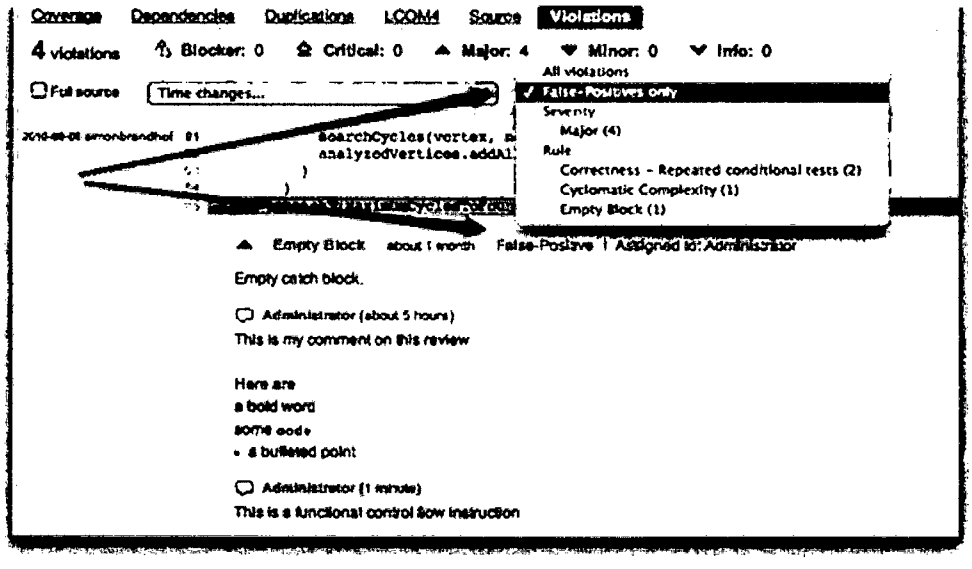


MARCAR UNA EVIDENCIA COMO FALSO POSITIVO

Si por alguna razón una evidencia se detecta como un falso positivo, es muy sencillo desactivarla marcándola como tal en la herramienta. Simplemente basta con pinchar en el enlace de "marcar como falso positivo":



Una vez ha sido desactivada, la evidencia no se volverá a mostrar más en el visor de recursos. Para verlos bastará con seleccionar la opción correspondiente en el filtro adecuado:



Además, todos los valores de las métricas en el proyecto como por ejemplo el número de evidencias encontradas será actualizado en la siguiente inspección de código.

5.1.19. USANDO SONAR EN ECLIPSE

PANORÁMICA



La API Sonar Eclipse proporciona una integración completa de Sonar en Eclipse para proyectos Java. El objetivo de esta integración es que los desarrolladores no dejan su IDE para gestionar la calidad de su código fuente. La mayoría de información que se muestra en la interfaz web Sonar ya está disponible en Eclipse.

Sonar Eclipse en la actualidad presta apoyo a tres casos de uso principales:

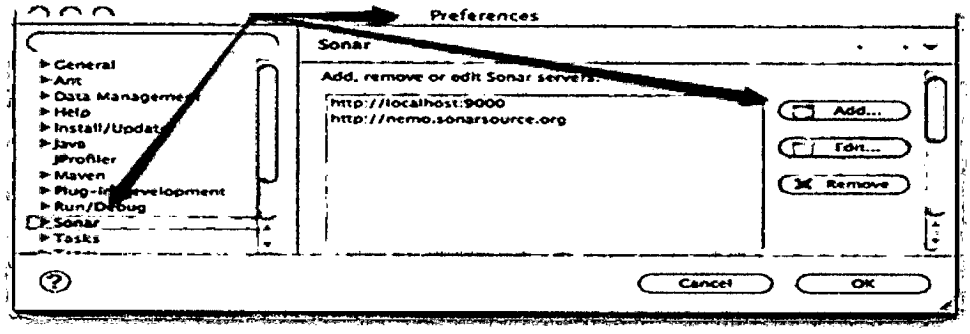
- Gestionar evidencias existentes.
- Trabajar a través de revisiones.
- Compruebe el código antes de cometer

CONFIGURANDO SONAR EN ECLIPSE

⚠ Para utilizar Sonar en Eclipse en un proyecto, se requiere que el proyecto ha sido analizado primero en una forma estándar por Sonar.

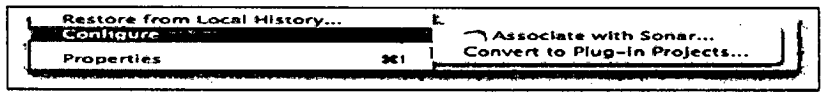
Establecer el Servidor Sonar

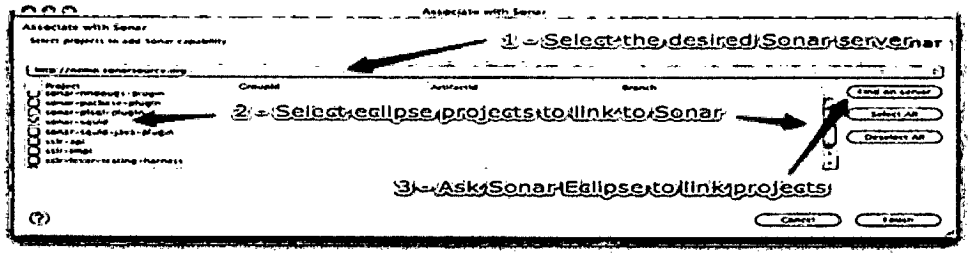
Sonar Eclipse está pre-configurado para acceder a un servidor local de Sonar mediante la dirección http://localhost:9000/. Para usar otra dirección, primero tiene que definir este servidor utilizando las Preferencias de Eclipse:



Vincular un proyecto al servidor Sonar

Una vez que el servidor Sonar está definido, el siguiente paso es vincular los proyectos de Eclipse con proyectos definidos y analizados en el servidor Sonar. Esta operación de unión se puede hacer haciendo clic en el botón derecho del ratón sobre el proyecto, seleccionando "Package Explorer", y luego en Configurar Asociado> Sonar con ...:

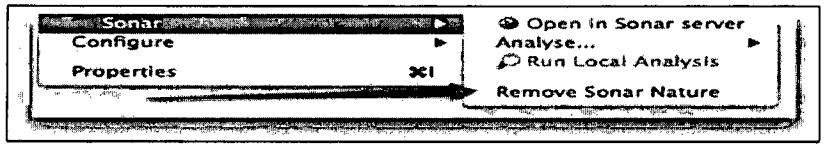




⚠ En algunos casos, el proyecto Eclipse no puede automáticamente ser asociado a un proyecto de Sonar. Si esto sucede, es necesario introducir manualmente la groupId y la artifactId del proyecto Sonar.
Por ejemplo, para el siguiente proyecto Sonar: http://nemo.sonarsource.org/dashboard/index/org.codehaus.sonar-plugins:sonar-flex-plugin, el groupId es "org.codehaus.sonar-plugins" y el artifactId es"sonar-flex-plugin".

Desvincular un Proyecto del Servidor Sonar

Esta operación se puede hacer haciendo clic en le botón derecho del ratón sobre el proyecto en el "Package Explorer", y luego Configuración> Eliminar Naturaleza Sonar:



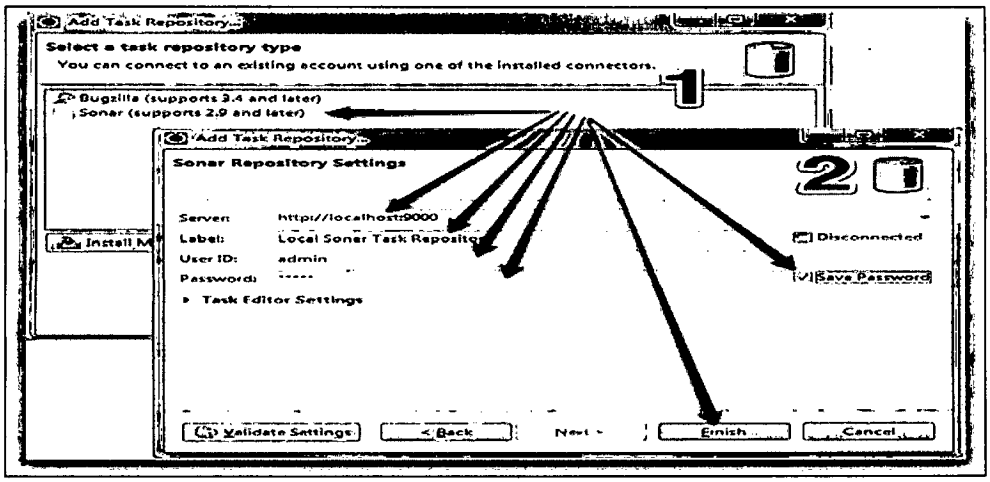
Configurando Mylin

Un conector Sonar Mylyn permite el seguimiento de revisiones.

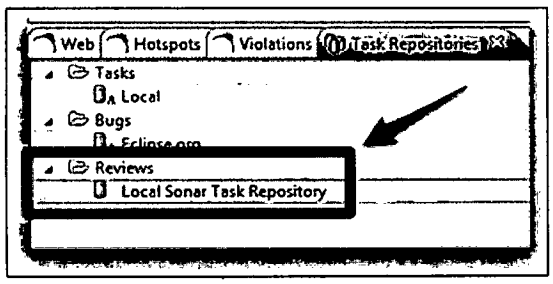
Creación de un repositorio de tareas Sonar

Para empezar a utilizar esta característica, se tiene que crear una nueva tarea en el repositorio de Sonar (vea la página de documentación Mylyn para más información):

⚠ **Limitaciones de la configuración de las validaciones**
Validar la configuración de las credenciales de autenticación, devuelve válidos incluso si no ha introducido las credenciales correctas (usuario / contraseña).
Por lo tanto, asegúrese de tratar correctamente las credenciales de lo contrario no será capaz de recuperar cualquier revisión.

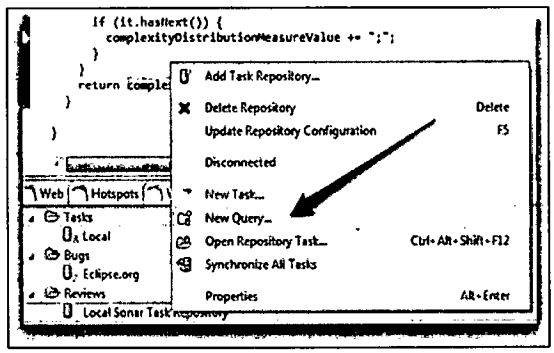


El nuevo repositorio debería aparecer ahora en la vista Repositorios de tareas:

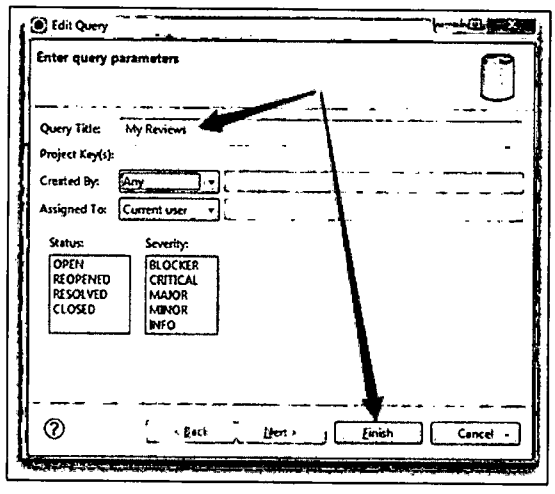


Creación de una Consulta

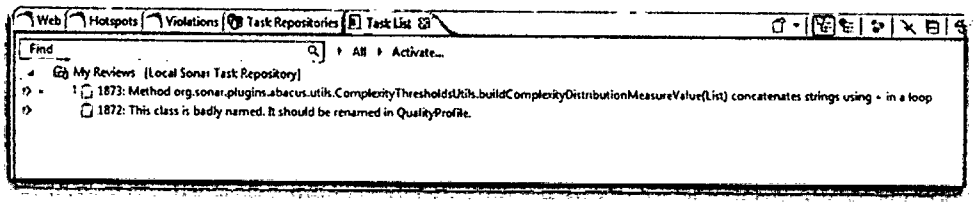
Para ver algunos de los comentarios, puede crear consultas tantas como desee:



He aquí un ejemplo de una consulta que recupera todas las opiniones que tiene asignadas en cada proyecto:



Los comentarios deben aparecer ahora en la vista Lista de tareas:



NAVEGADOR SONAR EN ECLIPSE

Modos Local y Remoto

Cuando se utiliza Sonar en Eclipse, tenemos disponibles dos modos:

Modo remoto

Este es el modo por defecto. La información mostrada en esta modalidad a distancia proviene de un servidor remoto de

Sonar. Esto significa que los datos se han calculado durante los últimos análisis en ese servidor.

Por lo tanto:

No podrá ver las evidencias introducidas en el código que acaba de agregar.

Las métricas (número de líneas por ejemplo) mostrarán los valores que no coinciden con lo que se ve en Eclipse si se ha añadido algo de código.

Modo Local

Cuando se cambia al modo local, la información que se muestra proviene de los últimos análisis realizados en local.

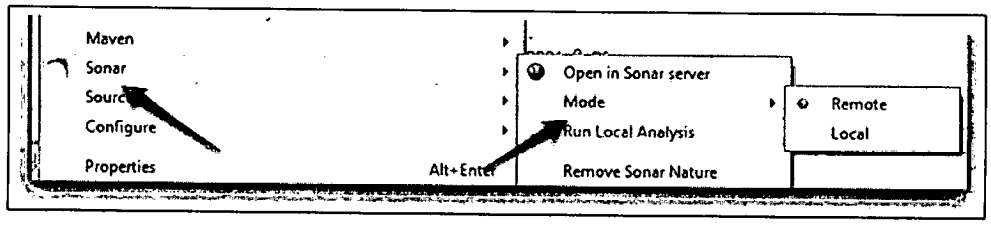
Por lo tanto:

Se pueden ver las evidencias introducidas en el código que acaba de agregar.

Las métricas (número de líneas por ejemplo) mostrará los valores que coinciden con lo que se ve en Eclipse si se ha añadido algo de código.

Cambiar de un Modo a Otro

Para cambiar de un modo a otro, haga clic en el proyecto, a continuación, seleccione Sonar> Mode> Control remoto o local:

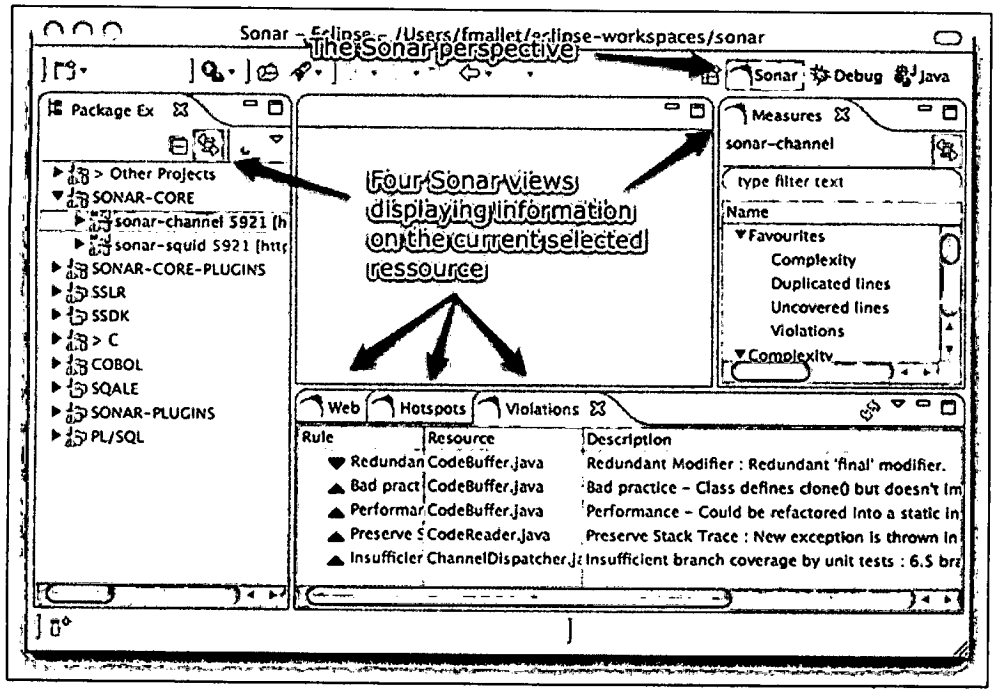


Las características serán marcados como remoto y / o local en función de su disponibilidad.

Vista de Sonar Eclipse

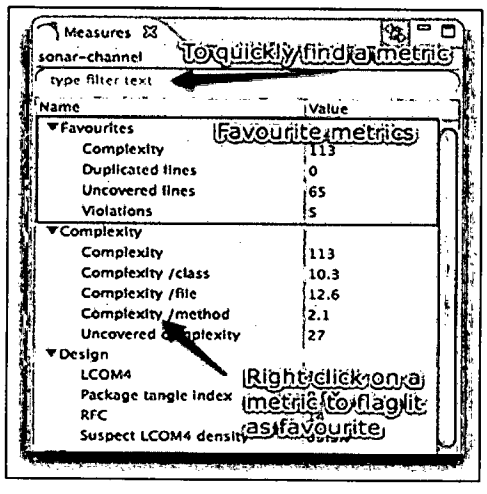
Sonar Eclipse ofrece una perspectiva de Eclipse y cuatro vistas de la herramienta. Estas vistas de Eclipse puedes usarse fuera de la perspectiva Sonar Eclipse, pero este tutorial describe el diseño predeterminado de la perspectiva de Sonar Eclipse.

Para activar la vista Sonar Eclipse, haga clic en Windows => Abrir perspectiva => Otros ... y seleccionar Sonar:



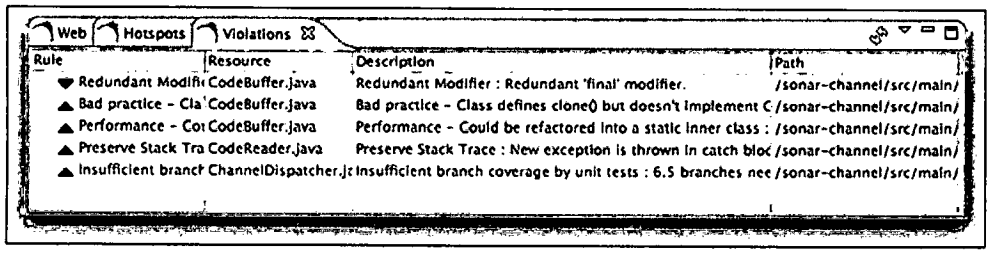
Vista de Métricas (Local o Remoto)

La vista de métricas permite reunir todas las medidas disponibles del recurso seleccionado. La mayoría de las veces, los desarrolladores centran su atención en sólo unos pocos parámetros como la complejidad, líneas código cubierto por las pruebas unitarias, las líneas duplicadas, etc. y necesita obtener una visión rápida de estas métricas. Por esta razón, las métricas predefinidas se muestran en la parte superior de la vista de las medidas. Se pueden añadir o eliminar las métricas de la lista de favoritos haciendo clic en el botón derecho del ratón sobre el nombre de métrica:



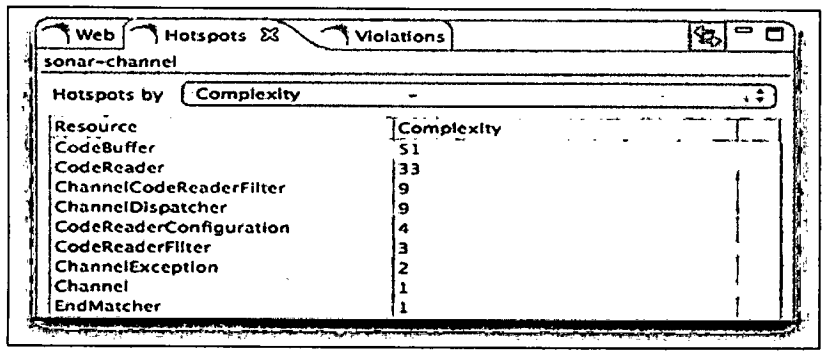
Vista de Evidencias (Local o Remoto)

La vista Evidencias muestra la lista de evidencias del recurso seleccionado (proyecto, archivo, etc.):



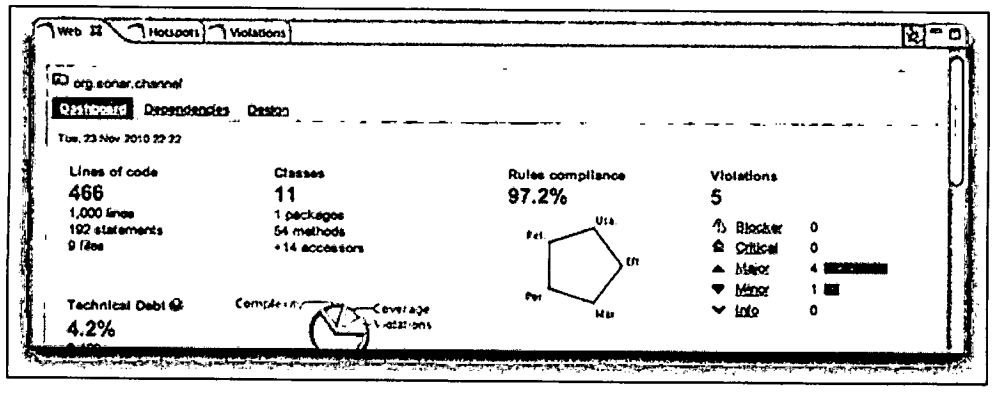
Vista Hotspots (Remoto)

La vista Hotspots permite identificar rápidamente los puntos de acceso a los archivos de acuerdo a las métricas definidas en favoritos (definida en la vista de medidas). Por ejemplo, puede ver los archivos con el mayor número de líneas duplicadas, el mayor número de evidencias, el mayor número de líneas no cubiertas por las pruebas unitarias, etc.



Vista Web (Remoto)

Esta vista muestra el panel de control Web de Sonar (para los proyectos y paquetes) o la vista web de Sonar para los recursos (para archivos). Esta página web se puede utilizar para acceder a información y servicios (nube, mapa de árbol,...) que aún no están disponibles de forma nativa en Sonar Eclipse:



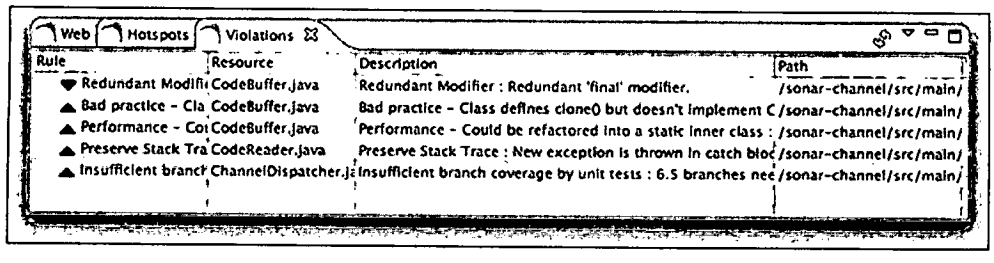
Trabajar con Sonar en Eclipse

Gestión de las Evidencias Existentes.

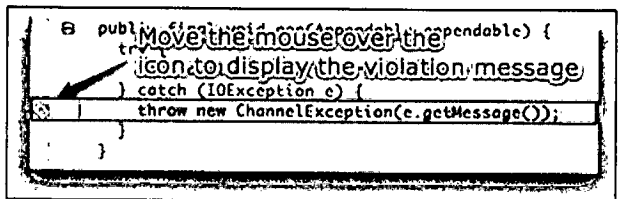
Al pasar el código fuente en el Sonar Eclipse se destacarán las evidencias existentes.

Detección de una Evidencia.

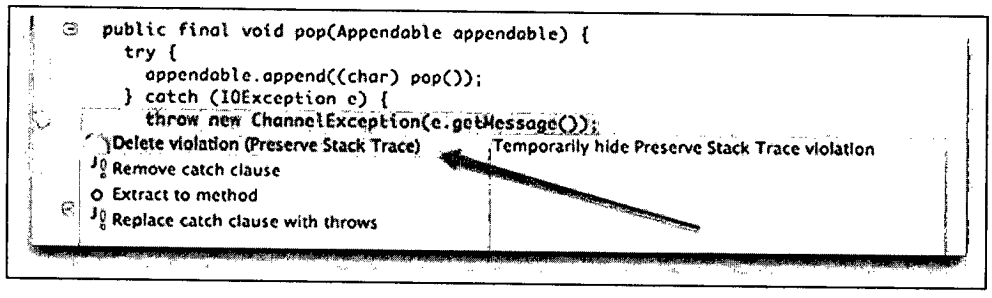
Mediante la vista de Evidencias se muestra la lista de evidencias del recurso seleccionado (proyecto, archivo, etc.):



Un doble clic sobre una evidencia abre el editor de código fuente y se destaca la evidencia:

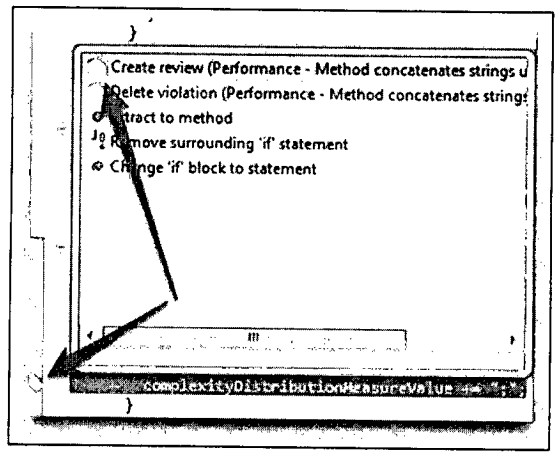


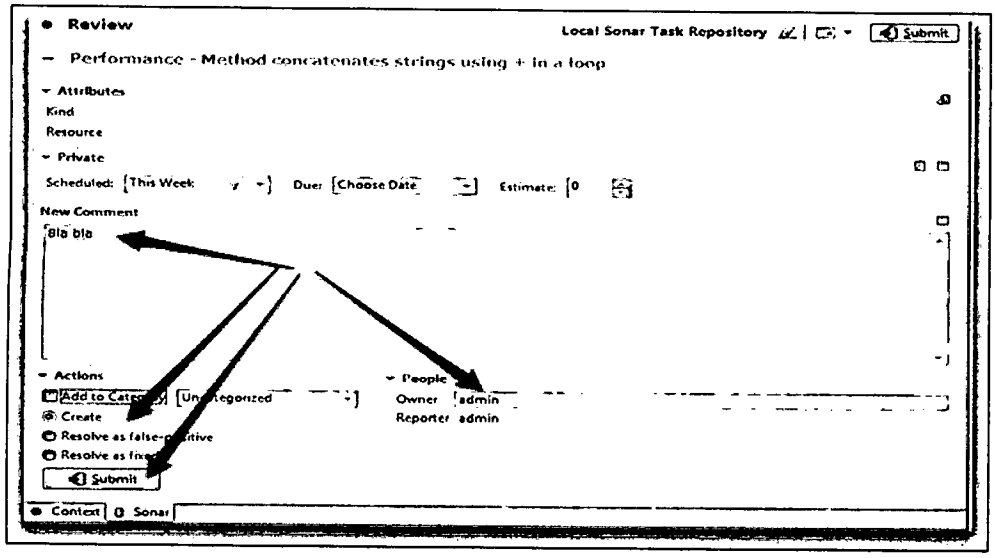
Una vez que una evidencia se ha solucionado, se elimina haciendo clic en el icono de evidencia y, a continuación, Eliminar. La eliminación de una evidencia también la quita de la vista Evidencias:



Creación de una Revisión (Remoto)

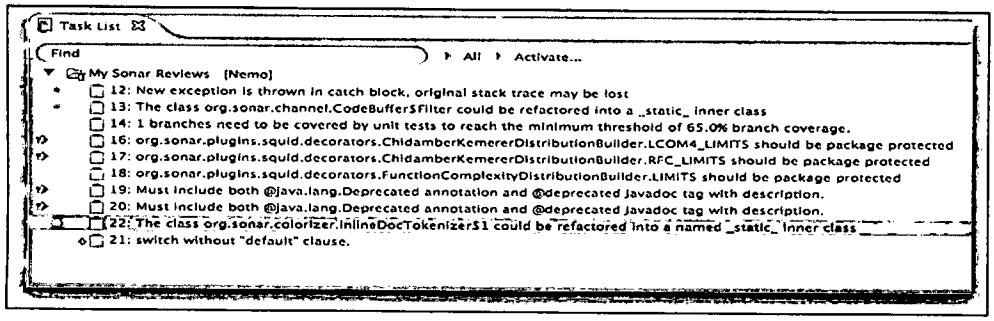
Si por alguna razón no se va a arreglar una evidencia de inmediato, pero cree que debe ser resuelta, puede crear una opinión sobre la misma.



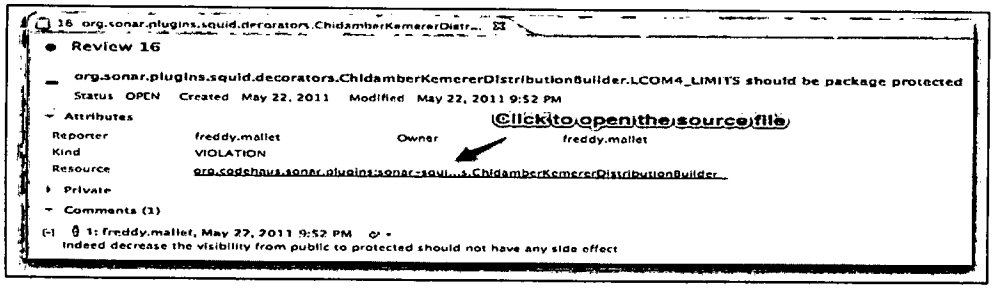


Trabajar Mediante las Revisiones

Es posible crear consultas personalizadas para mostrar los comentarios en la vista Lista de tareas:



A partir de ahí, hacer doble clic en una revisión abrir los detalles de dicha evaluación:



Todas las operaciones de reseñas disponibles a través de la interfaz web Sonar también están disponibles en Mylyn, por lo que una revisión puede ser comentada, fijada, reasignada, marcada como falso-positivo y abierta de nuevo. Además, una revisión se puede crear directamente desde el editor de código fuente de Eclipse en una evidencia existente.

⚠ Limitaciones conocidas

Esta extensión Mylyn cuenta actualmente con las siguientes limitaciones que serán rápidamente corregidos:

No existe una manera de crear una opinión que no se refiere a una evidencia existente

No existe una manera de crear una consulta Mylyn con filtros con el fin de, por ejemplo, recuperar únicamente opiniones sobre un proyecto, o revisiones asignadas a otro usuario

No existe una manera de añadir un comentario a un plan de acción: SONARIDE-286

Comprobación de código antes de hacer commit

Al agregar o cambiar el código de una aplicación, Sonar Eclipse ofrecen la posibilidad de mostrar la calidad de este código antes de incluirlo en la SCM.

⚠ Limitación conocida en los servicios diferenciales

Este caso de uso no es actualmente del todo compatible. Sonar Eclipse no proporciona un modo diferencial. Por ello no es posible diferenciar qué defecto existía antes de lo que ha sido añadido por los cambios.

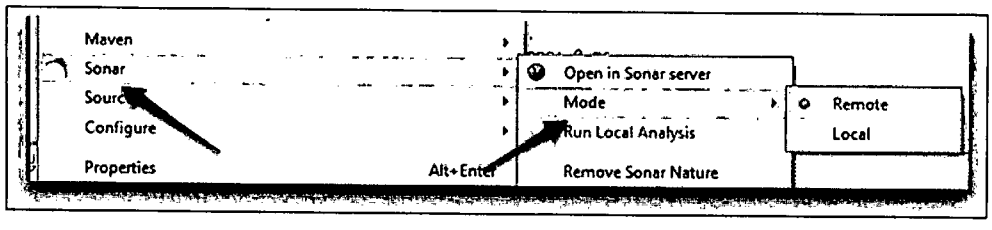
Análisis de Ejecución Local (Local)

Por defecto Sonar Eclipse se encarga de recopilar automáticamente la información desde el servidor Web Sonar. Este modo predeterminado es bastante útil, ya que no consume CPU o

la memoria en el ordenador del desarrollador mientras rápidamente mostrar información de valiosa calidad tan pronto como un nuevo archivo fuente se crea.

Pero si un montón de código fuente ha sido modificado a nivel local, la información de calidad disponible de forma remota puede estar muy desincronizado después de un tiempo. En ese caso, un análisis local se puede ejecutar antes de cualquier cambio en el repositorio de código fuente y con el perfil de calidad que se utilizó para analizar el proyecto de forma remota.

Para ejecutar un análisis local, se necesita cambiar al modo local. Tan pronto como se selecciona este modo se pone en marcha un análisis local para actualizar las evidencias y vistas de las métricas.



Posteriormente, en cualquier momento, se puede ejecutar un nuevo análisis local, haga clic en Sonar> Ejecutar Análisis Local.

Tenga cuidado
Incluso en el modo local, los puntos de acceso y puntos de vista Web siguen mostrando datos remotos.

CONCLUSIONES

1. Se ejecuto un estudio detallado para la respectiva certificación de codigo Java, donde se detectan vulnerabililidades en las aplicaciones webs que deben ser corregidas. En primera instancia se realiza una revisión general del codigo fuente, asimismo se describe el uso de la técnica para ejecutar el respectivo análisis, en cumplimiento de los estándares de calidad.

2. Existe un grado de asociación significativa entre las variables dependiente e independiente, tomando como referencia el cuestionario dirigido al equipo que elaboran proyectos de software de la consultora Everis, mediante el valor calculado o función prueba (Tc= 2.8466) y el valor de la tabla "T" student (T= 1.8125), aceptando la hipótesis alterna.

3. Se evaluo las métricas del Sonarqube 4.0 por: tamaño, pruebas, duplicidad, diseño, complejidad, reglas y SCM, para la verificación si los códigos cumplen con los requisitos establecidos y asi el análisis sea eficiente.

SUGERENCIAS

1. El encargado de los proyectos informáticos, debe realizar una propuesta para el análisis del codigo Java, dentro del ciclo de vida de desarrollo de los sistemas informaticos, con mayor énfasis en las metodologías agiles, ya que se alinean mejor a las métricas de evaluación del Sunarqube 4.0.

2. Los analistas de los proyectos informáticos, deben solicitar la aplicación del Sunarqube 4.0 en cada modulo implementado, para la respectiva fiabialidad en la certificación del codigo Java, asi evitar errores de codificación en la entrega del proyecto.

3. Los desarrolladores de los proyectos informáticos, deben realizar el análisis del codigo Java de manera continua, mediante la herramienta Jenkins, dado que realiza la integración automática y detecta fallos, siendo necesario para determinar las vulnerabilidades.

REFERENCIAS BIBLIOGRÁFICAS

Aguilera Díaz, R. (2014). *Analizando la calidad del código Java con Sonar.* Obtenido de https://www.adictosaltrabajo.com/tutoriales/utilizando-sonar/

Alba Castro, M. (2008). *Certificación automática de propiedades de seguridad de código fuente Java en lógica de reescritura.* Valencia - España: Universidad Politécnico de Valencia.

Alegsa, L. (05 de 12 de 2010). *Definición de código fuente.* Obtenido de http://www.alegsa.com.ar/Dic/codigo_fuente.php

Alvarez, M. (18 de 07 de 2001). *Descripción y características de este potente y moderno lenguaje de programación.* Obtenido de http://www.desarrolloweb.com/articulos/497.php

Bavera, F. (2006). *Compilación y certificación de código mediante análisis estático de flujo de control y de datos.* Montevideo - Uruguay: Universidad de la República.

Bernal Bermúdez, J. (2012). *Programación Orientada a Objetos con Java.* Madrid - España: Universidad Politécnica de Madrid.

Cambell, A. (2016). *Documentación SonarQube.* Obtenido de http://docs.sonarqube.org/display/SONAR/Documentation

Caro, F. (16 de 09 de 2010). *Sonar: Medida de la calidad de tu código.* Obtenido de https://www.paradigmadigital.com/dev/sonar-medida-de-la-calidad-de-tu-codigo/

Casañas, M. (2014). *Qué es software libre.* Obtenido de http://www.casanas.com.ar/attachments/Que_es_-_A_-_Conc_tecnicos.pdf

Cruz Vílchez, F. (2012). *Programación en Java.* España: Universitaria Andaluza Inca Garcilaso.

Dávila Murillo, P. (2010). *Software para controlar la calidad en el desarrollo de software SONAR.* Guayaquil - Ecuador: Universidad de Guayaquil.

Diccionario informático. (09 de 2011). *Diccionario para informáticos.* Obtenido de http://files.sld.cu/redenfermeria/files/2011/09/diccionario-informatico.pdf

García Oterino, A. (09 de 05 de 2014). *¿Qué es Jenkins?* Obtenido de http://www.javiergarzas.com/2014/05/jenkins-en-menos-de-10-min.html

Garcia, L. (03 de 12 de 2013). *¿Qué es SonarQube?* Obtenido de https://unpocodejava.wordpress.com/2013/12/03/que-es-sonarqube/

Gaudin, O. (29 de 11 de 2016). *Requisitos para SonarQube.* Obtenido de http://docs.sonarqube.org/display/SONAR/Requirements

Gigleux, A. (15 de 04 de 2016). *Arquitectura e Integración.* Obtenido de http://docs.sonarqube.org/display/SONAR/Architecture+and+Integration

Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, M. (2014). *Metodologia de la Investigacion.* México D.F: McGRAW - HILL.

IBM. (2015). *SPSS Statistics Base.* Obtenido de http://www-03.ibm.com/software/products/es/spss-stats-base

JavaSun. (2014). *Características de Java.* Obtenido de http://www.oracle.com/technetwork/java/index.html

Jenkins. (2014). *Jenkins: Construir grandes cosas a cualquier escala.* Obtenido de https://jenkins.io/

Kawaguchi Kohsuke. (30 de 01 de 2011). *Use Hudson: License.* Obtenido de http://hudson.gotdns.com/

Ospina Delgado, J. (2015). *Análisis de seguridad y calidad de aplicaciones (Sonarqube).* Manizales, Colombia: Universidad Obterta de Cataluyna.

Párraga García, I. (02 de 2003). *Curso de Java.* Obtenido de http://dis.um.es/~bmoros/privado/bibliografia/cursoJava.pdf

Pérez Porto, J., & Gardey, A. (2012). *Definición de código.* Obtenido de http://definicion.de/codigo/

Zamora Herrera, J. (2012). *Conceptos basicos.* Obtenido de https://mind42.com/mindmap/1c2144bd-212d-4169-92fc-2144e7c954b7?rel=pmb

ANEXOS

ANEXO 1. MATRIZ DE IDENTIFICACIÓN DEL PROBLEMA

N°	SÍNTOMAS	CONSECUENCIAS	PRONÓSTICO
01	Falta de normativas y protocolos	Duplicidad de codigo en las clases	Metodología de estandarización del codigo
02	Conflictos al momento de realizar la siguiente versión del software	Reestructuración total del código fuente en la siguiente versión del software	Evaluación continua del código fuente
03	Las clases y funciones no se encuentran debidamente relacionadas	Uso excesivo de variables locales	Reducción del número de variables locales
04	Clases con líneas de código extenso	Errores al momento de la compilación	Refactorización de clases o subclases para disminuir amplitud de código

ANEXO 2. HOJA DE RUTA DE LAS VARIABLES

PENSAMIENTO SISTEMICO

SonarQube 4.0 → Certificación de código Java

Definición de métricas Java por
Tamaño
Pruebas
Duplicidad
Diseño
Complejidad
Reglas
SCM

Dimensiones	Indicadores		
Métricas de software	1)	Funcionalidad	%
	2)	Confiabilidad	%
	3)	Usabilidad	%
	4)	Eficiencia	%
	5)	Eficacia	%
	6)	Portabilidad	%

Dimensiones	Indicadores		
Control de calidad	1)	Reportes	%
	2)	Verificación	%
	3)	Procesos	%
	4)	Recursos	%
	5)	Satisfacción	%

ANEXO 3. MATRIZ DE CONSISTENCIA

<table>
<tr><th>FORMULACIÓN</th><th>OBJETIVOS</th><th>HIPÓTESIS</th><th>METODOLOGÍA DE LA INVESTIGACIÓN</th></tr>
<tr>
<td><u>GENERAL:</u>
¿De qué manera SonarQube 4.0 mejora la certificación de código Java en una consultora de proyectos informáticos?

<u>ESPECÍFICOS:</u>

1.¿Cuál es el proceso de certificación de código Java en el ciclo de desarrollo de los proyectos informáticos para el cumplimiento de estándares de calidad?
2.¿Cuál es el grado de asociación entre SonarQube 4.0 y la certificación de código Java en una consultora de proyectos informáticos?
3.¿Cuál es la aplicación del SonarQube 4.0 como herramienta para la certificación de código Java en una consultora de proyectos informáticos?</td>
<td><u>GENERAL:</u>
Proponer el SonarQube 4.0 mejora la certificación de código Java en una consultora de proyectos informáticos .

<u>ESPECÍFICOS:</u>

1.Integrar la certificación de código Java en el ciclo de desarrollo de los proyectos informáticos para el cumplimiento de estándares de calidad.
2.Determinar el grado de asociación entre SonarQube 4.0 y la certificación de código Java en una consultora de proyectos informáticos.
3.Evaluar la aplicación del SonarQube 4.0 como herramienta para la certificación de código Java en una consultora de proyectos informáticos.</td>
<td><u>GENERAL:</u>
“SonarQube 4.0 mejora la certificación de código Java en una consultora de proyectos informáticos”.

<u>ESPECÍFICOS:</u>

1. “Existe integración de la certificación de código Java en el ciclo de desarrollo de los proyectos informáticos para el cumplimiento de estándares de calidad”.
2. “SonarQube 4.0 posee un grado de asociación con la certificación de código Java en una consultora de proyectos informáticos”.
3. “Es viable la aplicación del SonarQube 4.0 como herramienta para la certificación de código Java en una consultora de proyectos informáticos”.</td>
<td><u>TIPO DE INVESTIGACIÓN:</u>
Aplicada

<u>NIVEL DE INVESTIGACIÓN:</u>
Descriptivo

<u>MÉTODOS:</u>
Inductivo – Deductivo.

<u>DISEÑO DE LA INVESTIGACIÓN:</u>
Pre Test – Post Test

<u>POBLACIÓN & MUESTRA:</u>
16

<u>SW DE PROC. DATOS:</u>
EXCEL

<table>
<tr><td>Fuentes</td><td>Técnicas</td><td>Instrumentos</td></tr>
<tr><td>Primaria</td><td>Encuesta</td><td>Cuestionario</td></tr>
<tr><td>Secundaria</td><td>Analisis documental</td><td>Fichas textuales y de resumen</td></tr>
</table>
</td>
</tr>
</table>

ANEXO 4. CUESTIONARIO DIRIGIDO AL EQUIPO QUE ELABORAN PROYECTOS DE SOFTWARE DE LA CONSULTORA EVERIS





"La aplicación de la encuesta busca determinar la influencia del SonarQube 4.0 en la mejora la certificación de código Java en una consultora de proyectos informáticos".

Numero de cuestionario:

Fecha:

Codigo de encuestador:

Señores a continuación se presenta un cuestionario para su desarrollo. Usted debe señalar con un ASPA (X) que tanto está de acuerdo con las preguntas, marcando correctamente dentro del recuadro.

PREGUNTAS ESPECIFICAS

1	2	3	4	5
Muy malo	**Malo**	**Regular**	**Adecuado**	**Bueno**

N°	**PREGUNTAS**	**ALTERNATIVAS**				
	SONARQUBE 4.0.					
P01	¿Cómo califica la funcionalidad del SonarQube 4.0 en la certificación de código Java?	1	2	3	4	5
P02	¿Cómo califica la confiabilidad del SonarQube 4.0 en la certificación de código Java?	1	2	3	4	5
P03	¿Cómo califica la usabilidad del SonarQube 4.0 en la certificación de código Java?	1	2	3	4	5

P04	¿Cómo califica la eficiencia del SonarQube 4.0 en la certificación de código Java?	1	2	3	4	5
P05	¿Cómo califica la eficacia del SonarQube 4.0 en la certificación de código Java?	1	2	3	4	5
P06	¿Cómo califica la portabilidad del SonarQube 4.0 en la certificación de código Java?	1	2	3	4	5
CERTIFICACIÓN DE CÓDIGO JAVA						
P07	¿Cómo califica los reportes emitidos del SonarQube 4.0 en la certificación de código Java?	1	2	3	4	5
P08	¿Cómo califica la verificación del SonarQube 4.0 en la certificación de código Java?	1	2	3	4	5
P09	¿Cómo califica los procesos del SonarQube 4.0 en la certificación de código Java?	1	2	3	4	5
P10	¿Cómo califica los recursos del SonarQube 4.0 en la certificación de código Java?	1	2	3	4	5
P11	¿Cómo califica la satisfacción del SonarQube 4.0 en la certificación de código Java?	1	2	3	4	5

Gracias por su colaboración

ANEXO 5. VALIDACIÓN DEL INSTRUMENTO

CONFIABILIDAD Y VALIDEZ DEL INSTRUMENTO DE MEDIDA.

El Alfa de Cronbach, evalúa la fiabilidad consistente de los datos. Dicho coeficiente examina la uniformidad, es decir, la cantidad de datos ingresados que se compone mediante una escala. Asimismo conocemos que el valor se encuentra entre 0 y 1, para considerar que los datos no se encuentren dispersos y exista correlación el Alfa de Cronbach debe ser como mínimo mayor o igual a 0.7, es importante señalar que a medida que mejore el coeficiente, existe mayor confiabilidad de los datos.

ANÁLISIS DE FIABILIDAD.

Resumen del procesamiento de los casos

		N	%
Casos	Válidos	16	100,0
	Excluidos[a]	0	,0
	Total	16	100,0

a. Eliminación por lista basada en todas las variables del procedimiento.

Estadísticos de fiabilidad

Prueba	Alfa de Cronbach	N de elementos
Pre Test	0,844	11
Post Test	0,822	11

ESTADÍSTICOS TOTAL DE ELEMENTOS (PRE – TEST)

N°	Media de la escala si se elimina el elemento	Varianza de la escala si se elimina el elemento	Correlación elemento -total corregida	Alfa de Cronbach si se elimina el elemento
P01	18,6875	22,363	,671	,819
P02	18,8750	25,317	,327	,846
P03	18,9375	25,263	,399	,840
P04	18,4375	22,929	,589	,826
P05	18,6875	22,096	,710	,815
P06	18,5000	22,800	,564	,828
P07	18,8125	22,296	,715	,815
P08	18,5625	26,396	,127	,864
P09	18,6875	23,563	,580	,827
P10	18,8125	22,296	,631	,822
P11	18,6250	24,383	,489	,834

ESTADÍSTICOS TOTAL DE ELEMENTOS (POST – TEST)

N°	Media de la escala si se elimina el elemento	Varianza de la escala si se elimina el elemento	Correlación elemento -total corregida	Alfa de Cronbach si se elimina el elemento
P01	42,6250	20,117	,406	,819
P02	42,5000	20,267	,523	,805
P03	42,3125	17,829	,864	,768
P04	42,3750	21,183	,508	,807
P05	42,7500	20,333	,493	,807
P06	42,5000	18,800	,599	,796
P07	42,1875	20,563	,593	,800
P08	42,5625	20,929	,443	,812
P09	42,3750	20,650	,503	,807
P10	42,3750	22,517	,356	,819
P11	42,3125	22,229	,195	,836

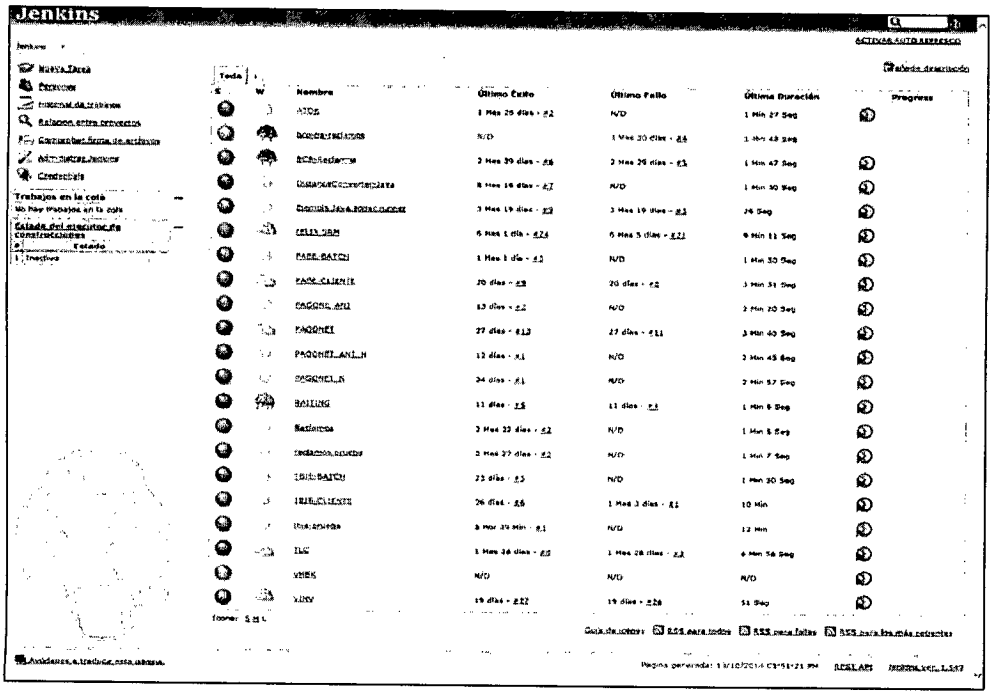
ANEXO 6. MANUAL DE USUARIO JENKINS

1. ¿Cómo acceder a Jenkins?

Para acceder a Jenkins lo primero será acceder a la siguiente dirección URL:

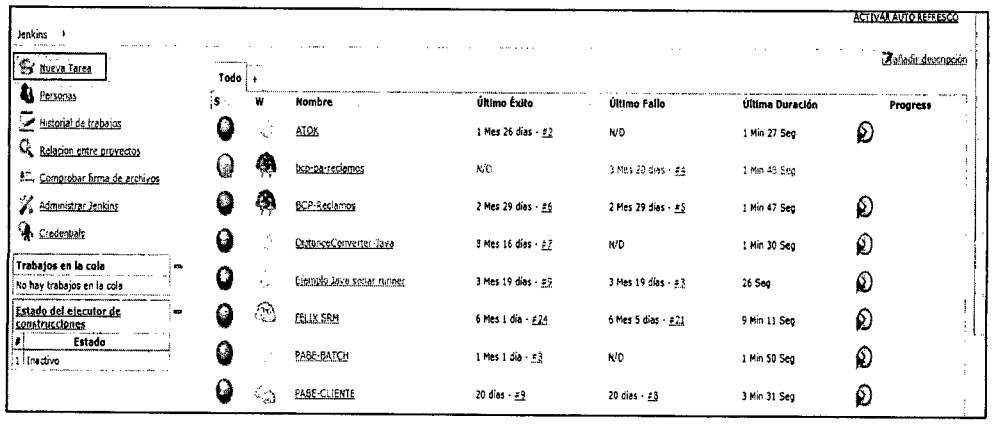
http://10.227.8.15:8080/

Una vez ingresado a esta dirección URL aparecerá la siguiente ventana:

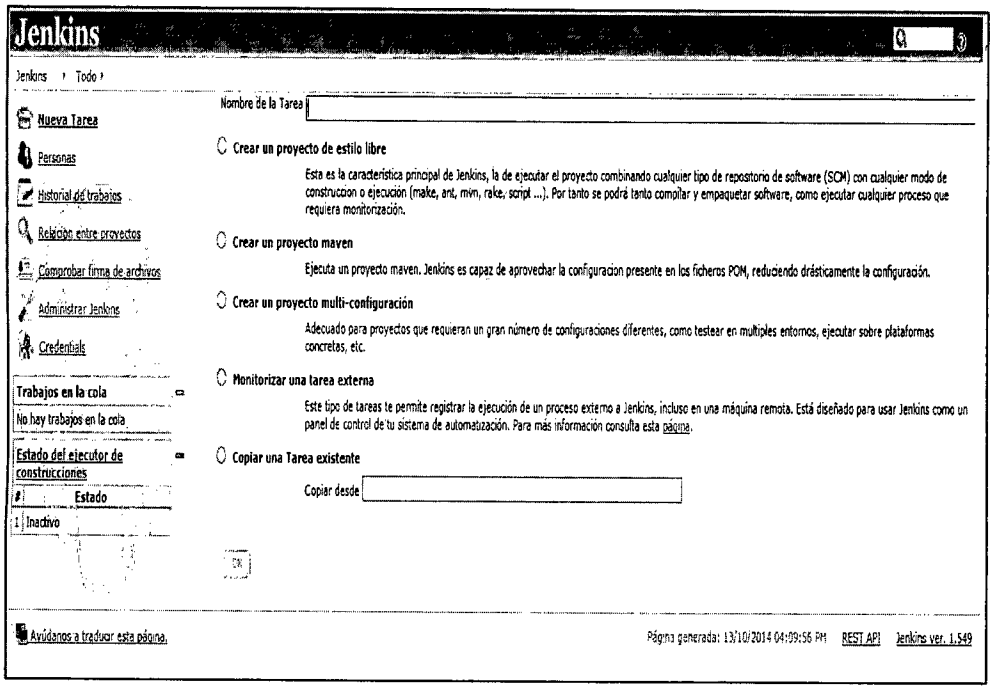


1.1. Nueva tarea

Para crear una nueva tarea el usuario deberá dar clic en la opción "Nueva Tarea" tal como se muestra en la imagen posterior.

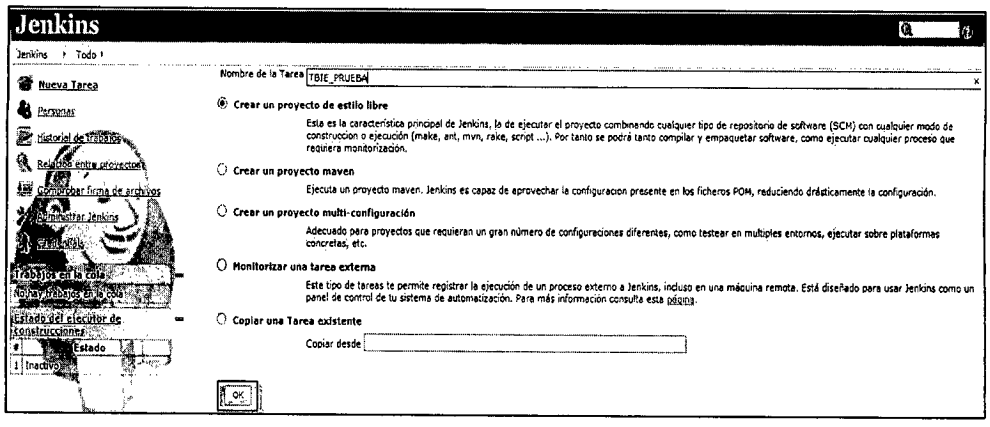


Luego de haber dado clic en la opción "Nueva Tarea" aparecerá la siguiente ventana:

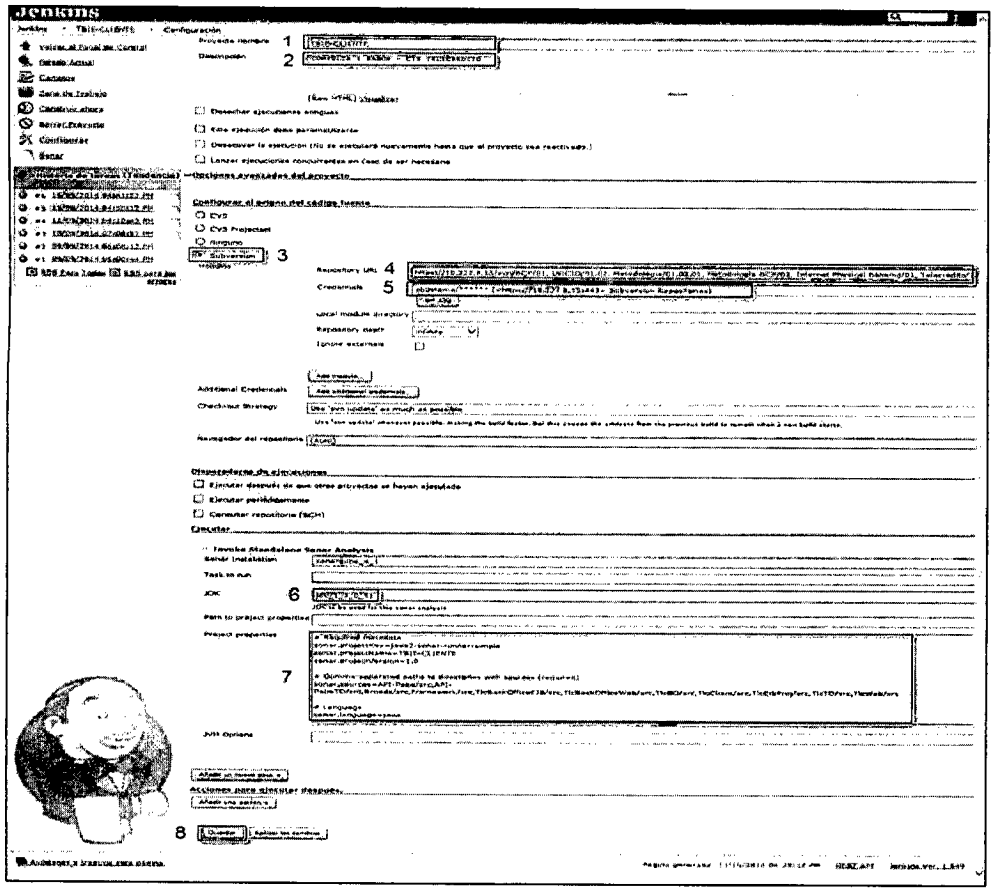


Para iniciar una nueva tarea debemos registrar el nombre de la tarea y seleccionar el tipo de proyecto a emplear (Crear un proyecto de estilo libre, Crear un proyecto maven, Crear un proyecto multi – configuración, Monitorear una tarea externa, Copiar una tarea existente). Para este

caso seleccionar la opción "Crear un proyecto de estilo libre" y dar clic en "OK"



Luego de haber dado clic en la opción anterior se abrirá la siguiente ventana donde se realizaran los siguientes pasos:



1. En esta sección el usuario registrara el nombre del proyecto tal como se muestra en la figura siguiente (1).

2. ingresara la descripción del proyecto (2).

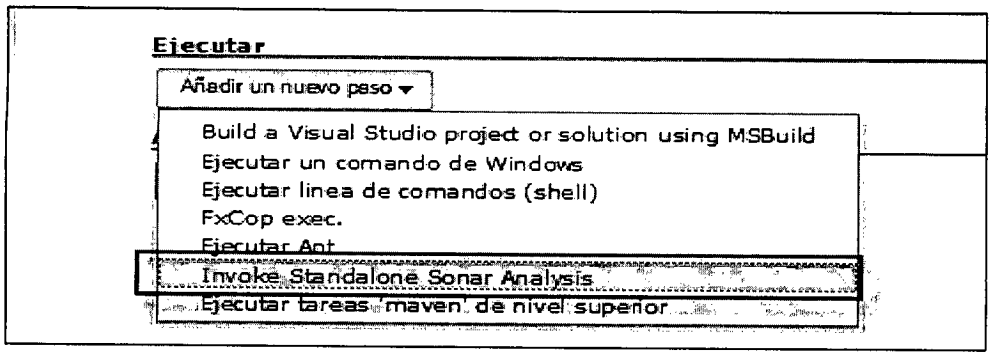
3. En la sección Configurar el origen del código fuente, marcar la opción Subversión (3).

4. Ingresar la URL del repositorio, esta es la ruta de origen donde se encuentra el proyecto, ejemplo (4).

   https://10.227.8.15/svn/BCP/01. INICIO/01.02. Metodologia/01.02.01. Metodologia BCP/02. Internet Physical Banking/01. Telecredito/1.Documentacion/Archivos Class_Sonar/TBIE/TLC

5. Seleccionar la credencial del proyecto (5).

   En la opción ejecutar en la opción añadir un nuevo paso, seleccionar la opción "Invoke Standalone Sonar Analisys"



6. Una vez seleccionada esta opción ingresar el JDK del proyecto, para este caso la versión a utilizar será el Jdk 1.7 (6).

7. En la sección Project properties copiar el siguiente código:

```
//Texto a colocar

# Required metadata

sonar.projectKey=java2-sonar-runner-simple

sonar.projectName=TBIE-CLIENTE

sonar.projectVersion=1.0

# Comma-separated paths to directories with sources (required)

sonar.sources=API-Pabe/src,API-
PabeTO/src,Broads/src,Framework/src,TlcBackOfficeEJB/src,TlcBac
kOfficeWeb/src,TlcBO/src,TlcClient/src,TlcEjbProj/src,TlcTO/src,Tlc
Web/src

# Language

sonar.language= java

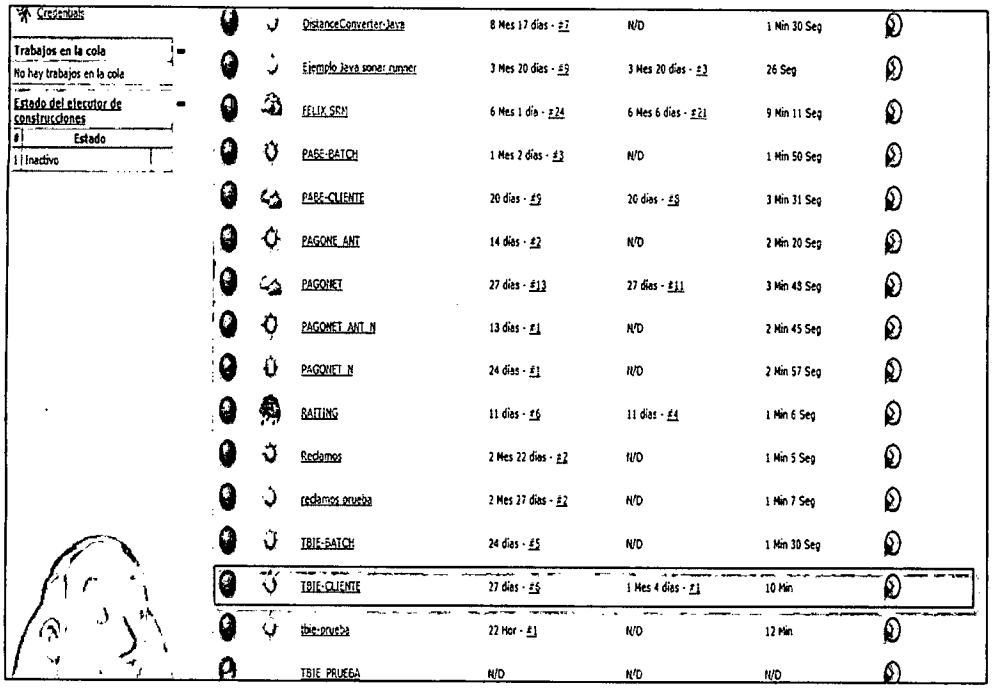
//
```

Las líneas marcadas son editables por ejemplo el projectKey es el key del proyecto cada key es único e irrepetible si se coloca un key repetido esto generaría que se reemplace el key anterior y se altere en los resultados.

El projectName es el nombre del proyecto este campo también será editado y aparecerá en la vista de resultados con el nombre que se le indique.

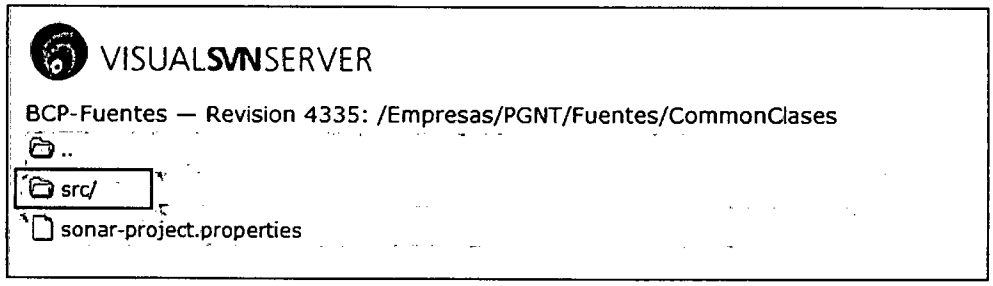
Ejemplo

sonar.projectName=TBIE-CLIENTE



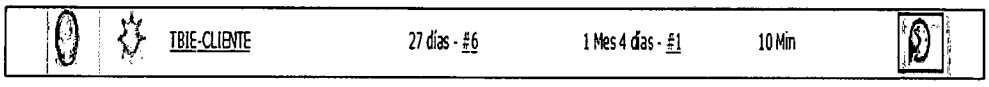
Y por último se edita el "sonar.sources" indicando el paquete de fuentes

Nota: Empaquetar los elementos de fuentes si es que no contiene el directorio source o la carpeta src, para empaquetar los elementos deberá generar una carpeta con el nombre "src/" e copiar todos los elementos del proyecto dentro de la carpeta src/ tal como se muestra a continuación:



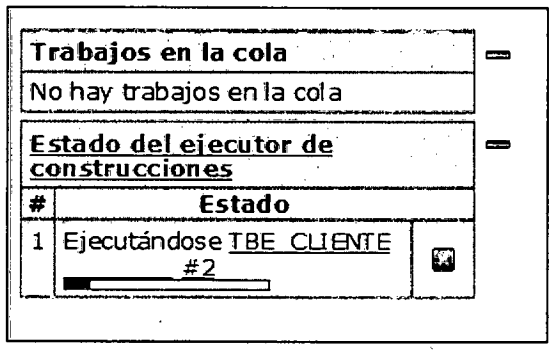
8. Para finalizar dar clic en el botón "Guardar".

Luego de haber realizado los pasos para la configuración en el punto anterior, usted podrá visualizar la lista de proyectos a ejecutar entre los cuales visualizara el proyecto generado tal como se muestra a continuación en la siguiente imagen:

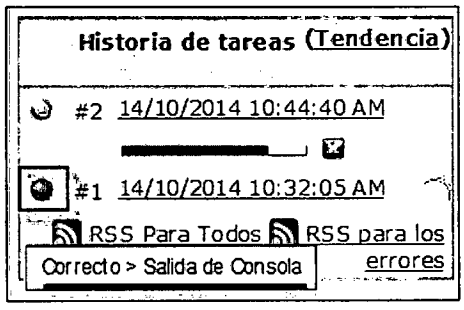


Una vez localizado el proyecto generado dar clic en el botón "Programar una construcción" ( ).

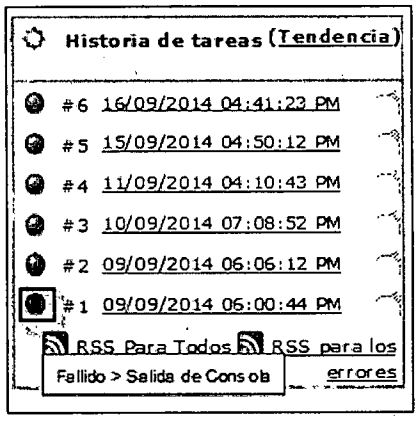
Y a continuación en el panel izquierdo en la sección "Estado del ejecutor de construcciones" se observara en el estado que se está ejecutando la barra de tareas del proyecto.



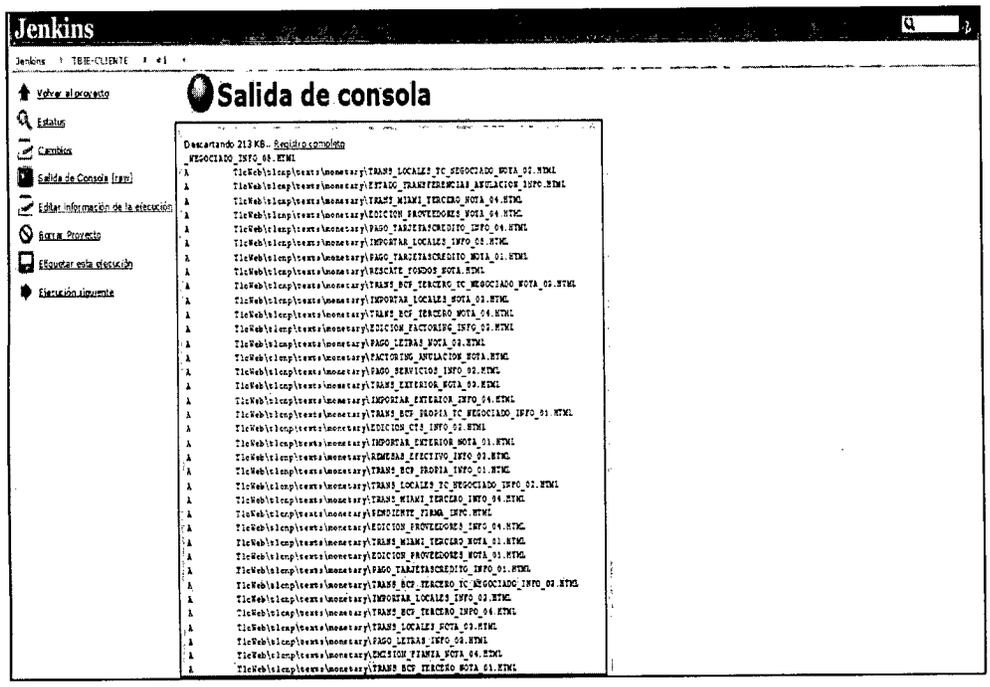
Cuando la tarea se ejecutó correctamente al costado de la tarea se observa el icono de color azul ( ) esto nos indica que la configuración se realizó de forma correcta es decir la salida de consola está en estado correcto tal como se muestra en la imagen siguiente:



De no ser así el icono que aparecerá será de color rojo ( ) esto nos indica que hubieron fallas en la configuración, es decir se detectaron fallas en la salida de la consola, de ser así se deberá revisar la consola de error para acceder a la consola usted deberá dar clic en el icono ( ) tal como se muestra en la siguiente imagen:



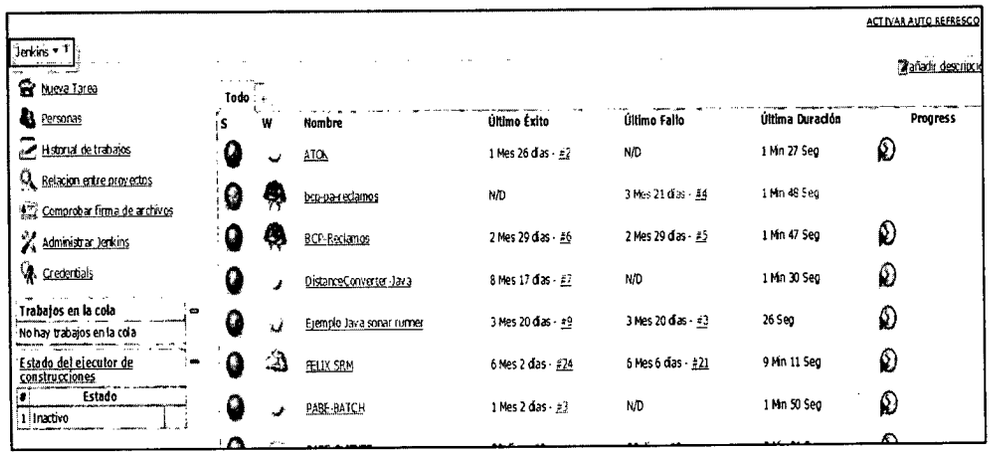
Una vez dado clic se mostrara la ventana salida de consola tal como se muestra a continuación en la siguiente imagen, es en esta ventana donde se muestran los errores por los que la consola de Jenkins no puede ejecutar de forma correcta.



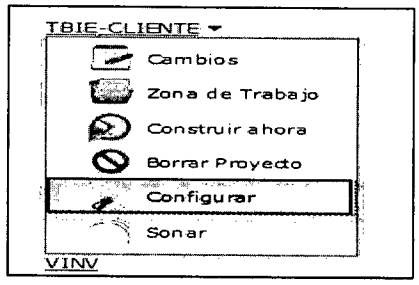
Una vez que se haya ejecutado correctamente la tarea se procederá a ingresar al Sonar para continuar con los pasos siguientes a esta configuración.

1.2. Editar tarea

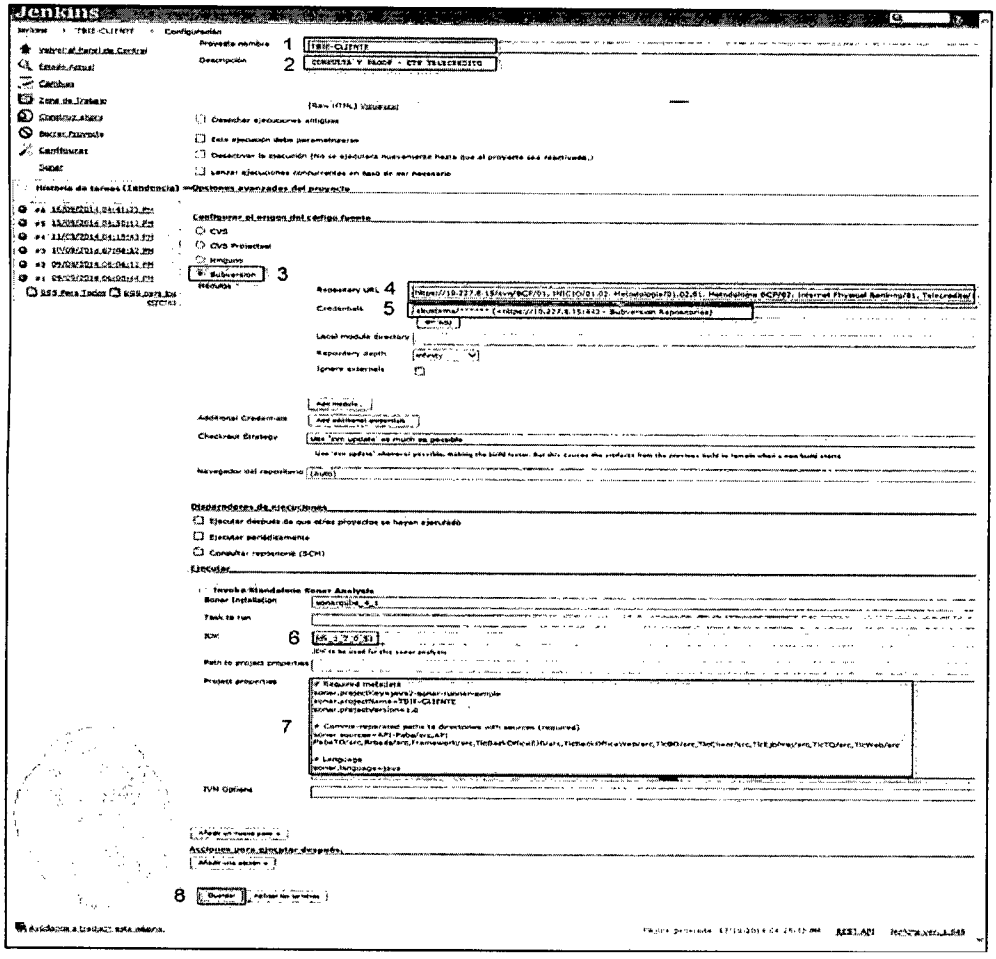
Para editar una tarea acceder a la lista de proyectos para ello dar clic en la pestaña "Jenkins" y se mostrara la siguiente ventana:



Dar clic en el nombre del proyecto a configurar y desplegar el icono (▾) y seleccionar la opción "Configurar"

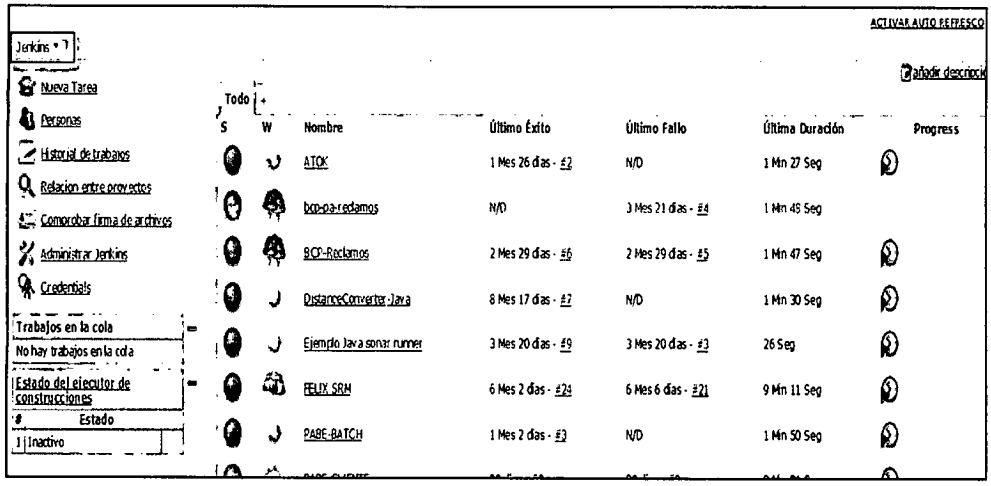


Una vez dado clic en la opción "Configurar" se mostrara la siguiente ventana, donde se podrá realizar las configuraciones y cambios respectivos.

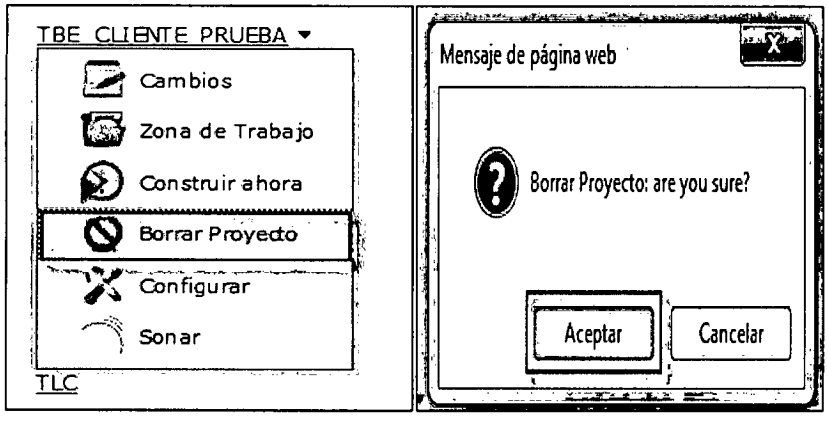


1.3. Eliminar tarea

Para eliminar una tarea acceder a la lista de proyectos para ello dar clic en la pestaña "Jenkins" y se mostrara la siguiente ventana:



Una vez localizado el proyecto dar clic en el icono (▾) y seleccionar la opción "Borrar Proyecto" una vez hecho el paso anterior aparecerá una ventana con el siguiente mensaje "Borrar Proyecto, are you sure", para eliminar el proyecto dar clic en el botón "Aceptar" de lo contrario dar clic en "Cancelar".



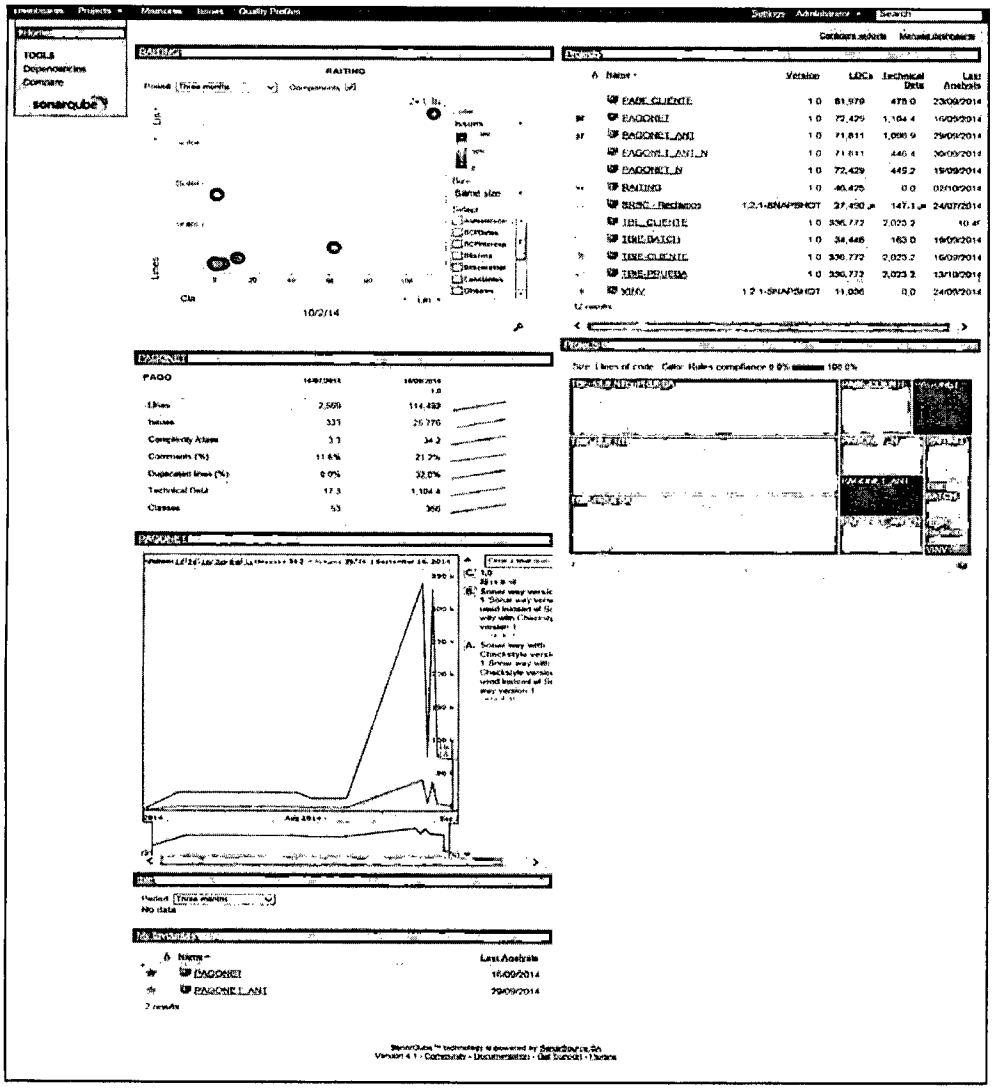
ANEXO 7. MANUAL DE USUARIO SONARQUBE

1. ¿Cómo acceder a SonarQube?

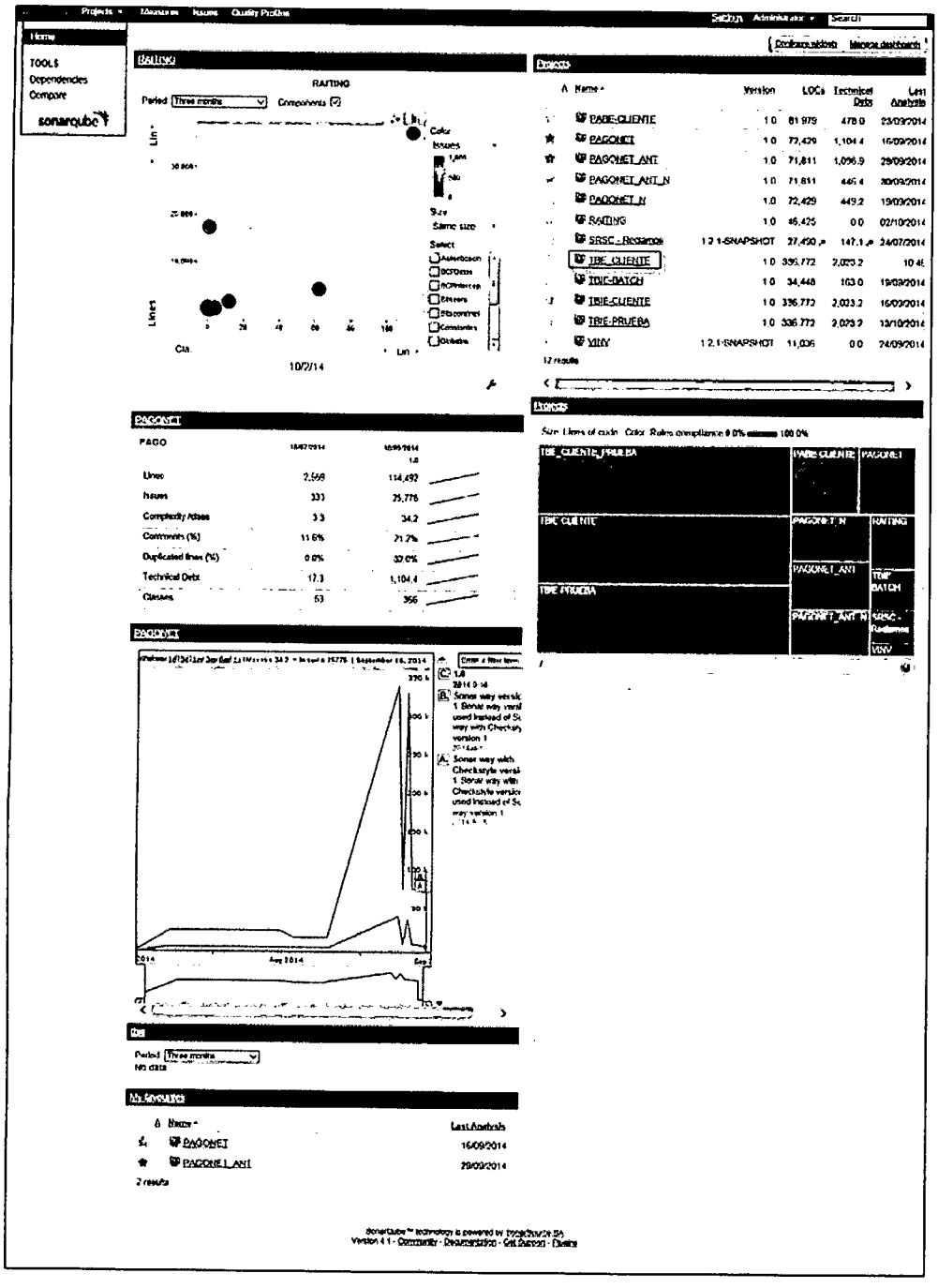
Para acceder a SonarQube lo primero será acceder a la siguiente dirección URL:

http://10.227.8.15:9000/

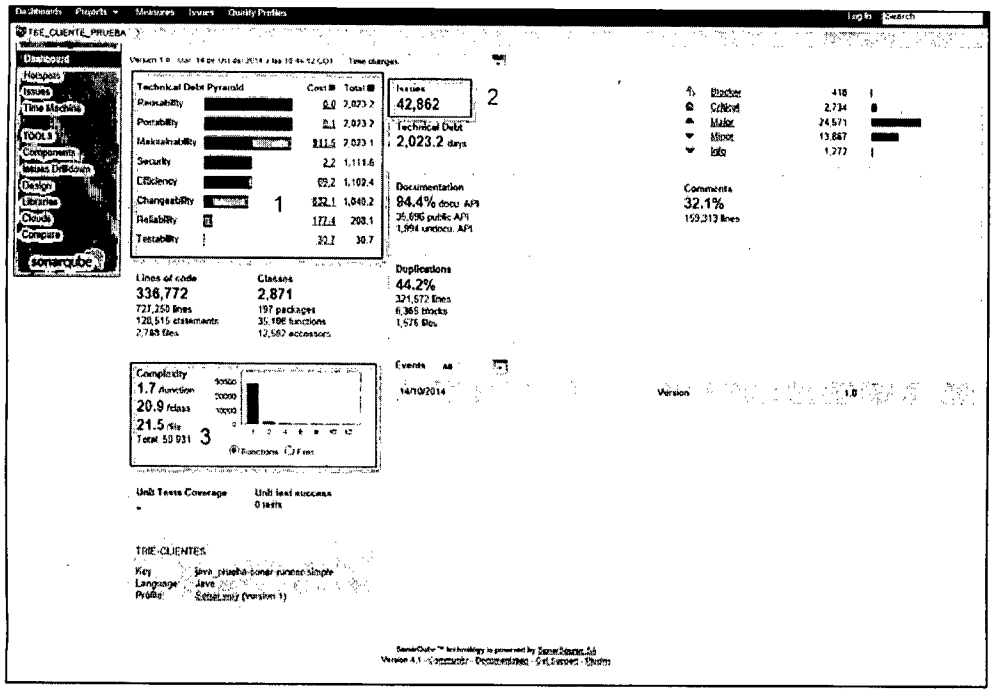
Una vez ingresado a esta dirección URL aparecerá la siguiente ventana:



Una vez visualizado la siguiente pantalla en la sección "Projects" se observa una lista de proyectos desplegados en el sonar, entre ellos buscar el nombre del proyecto generado en el Jenkins tal como se muestra a continuación en la siguiente imagen:



Para tener mas informacion precisa sobre el estado del projecto se dara clic en el nombre del projecto y aparecera la siguiente ventana(Dashboard):



Technical Debt Pyramid

En esta sección se muestra un reporte del estado en el que se encuentra el proyecto según la evaluación realizada por el Sonar, entre ellos tenemos el grafico de la deuda técnica (Technical Debt Pyramid) con los siguientes caracteristicas:

- Reusability: En este grafico de barras se muestra un análisis realizado por sonar donde se observa la capacidad que tiene el programa para reutilizar su código de manera eficiente, rápida y sencilla adaptable a cambios sin que estos afecten a otras clases ni ponga en peligro la estabilidad del sistema

- Portability: En este grafico de barras se observa la capacidad del programa para ser transferido desde una plataforma a otra permitiendo la comunicación entre estas.

- Maintainability: En este grafico de barras se muestra los atributos que tienen facilidad de extender, modificar o corregir errores del sistema.

- Security: este grafico de barras tiene como finalidad mostrar los atributos que tiene el programa respecto en cuanto a la seguridad para prevenir el acceso a personas no autorizadas (permisos, accesos, etc.) al sistema y ni el acceso a datos e información confidencial.

- Efficiency: este grafico tiene como finalidad medir el desempeño del software y la cantidad de recursos necesitados bajo las condiciones establecidas, respetando ciertos estándares.

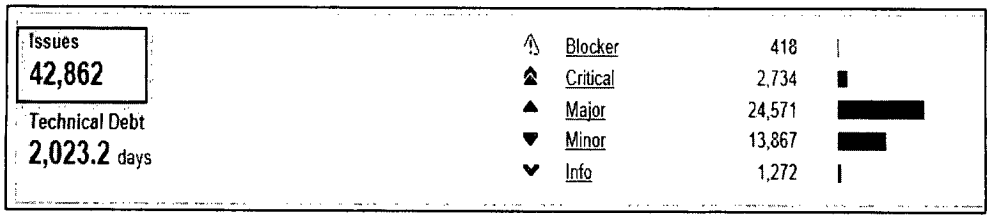
- Changeability: este grafico tiene como finalidad medir la facilidad que presenta el aplicativo en cuanto sea sujeto a modificaciones en las líneas de código.

- Reliability: el grafico mostrado indica el nivel de confiabilidad que presenta el aplicativo en cuanto al entorno de desarrollo es decir la tolerancia a fallos y su capacidad de recuperación.

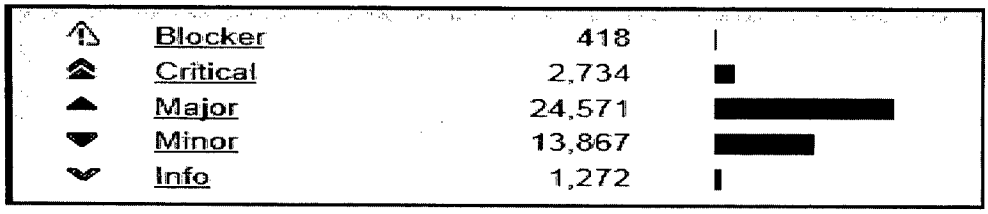
- Testability: en el siguiente grafico se muestra si el sistema se está aplicando el uso de las herramientas de testeo tales como Junit, Xunit, entre otros.

Issues

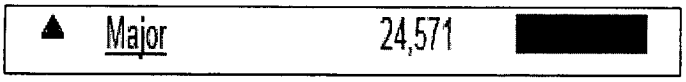
En esta sección se muestra la cantidad de errores de todo el aplicativo.



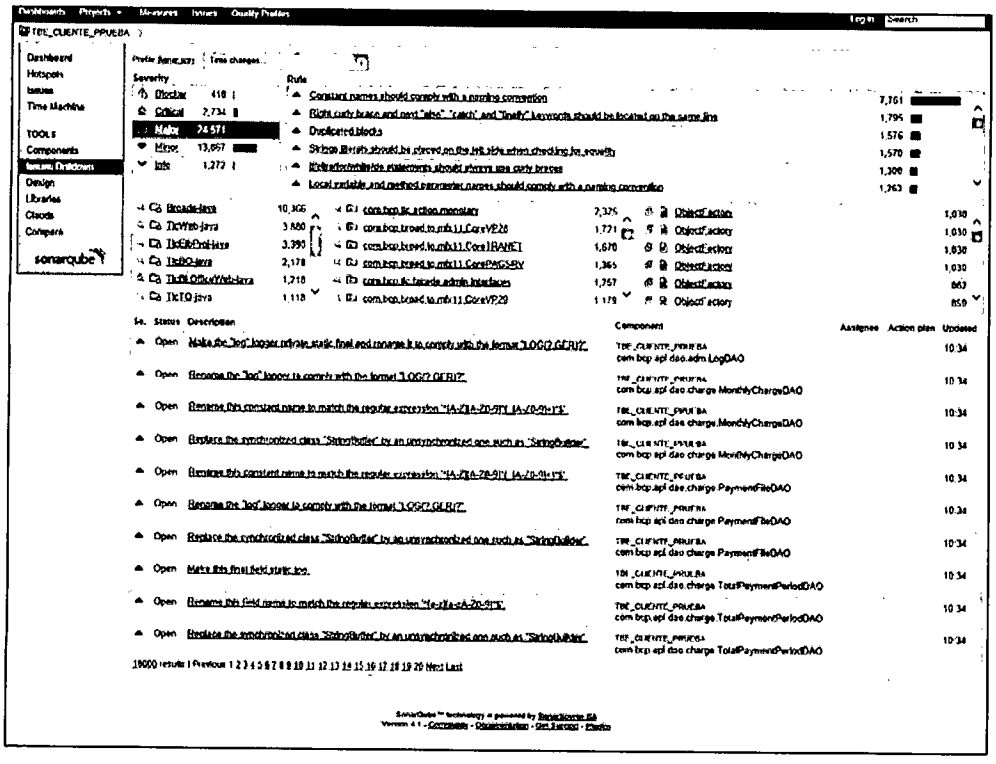
Para visualizar los errores de programación detectados por el Sonar se debe ir a la lista de clasificación de errores donde se clasificaran los errores de acuerdo al tipo de priorización mostrados a continuación en la siguiente imagen:



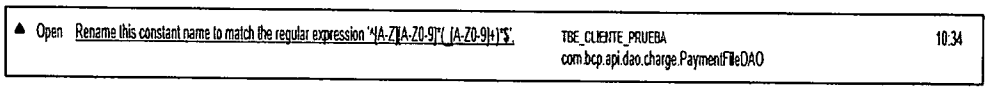
Para visualizar los errores de programación se deberá dar clic en uno de los elementos de la lista de priorización



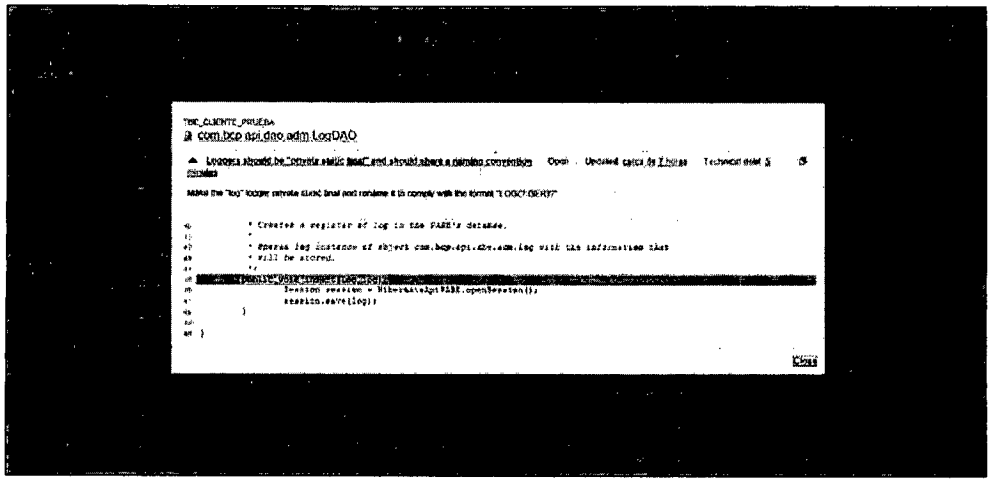
Una vez realizado el punto anterior se muestra la ventana siguiente.



Para visualizar el error de líneas de código se debe dar clic en el campo de la descripción del error tal como se muestra en la imagen siguiente.

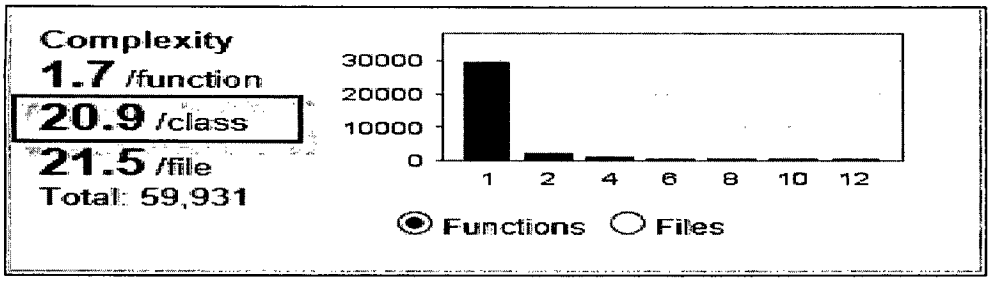


Luego se abrirá la una ventana donde se muestra las líneas de código que el Sonar detecto con observación o presencia de error y las posibles recomendaciones a manera de ayuda.



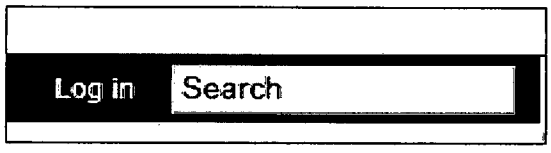
Complexity

En esta sección se puede observar el promedio de la complejidad de las clases o también conocida como la complejidad ciclomática con este resultado podemos indicar cuanta complejidad se presenta en la codificación del aplicativo por ejemplo que un aplicativo tenga demasiadas estructuras de control (if, else, while, etc) que el código no sea reutilizado es decir que utilice código repetido pudiendo heredar clases o reutilizar controles entre otros.

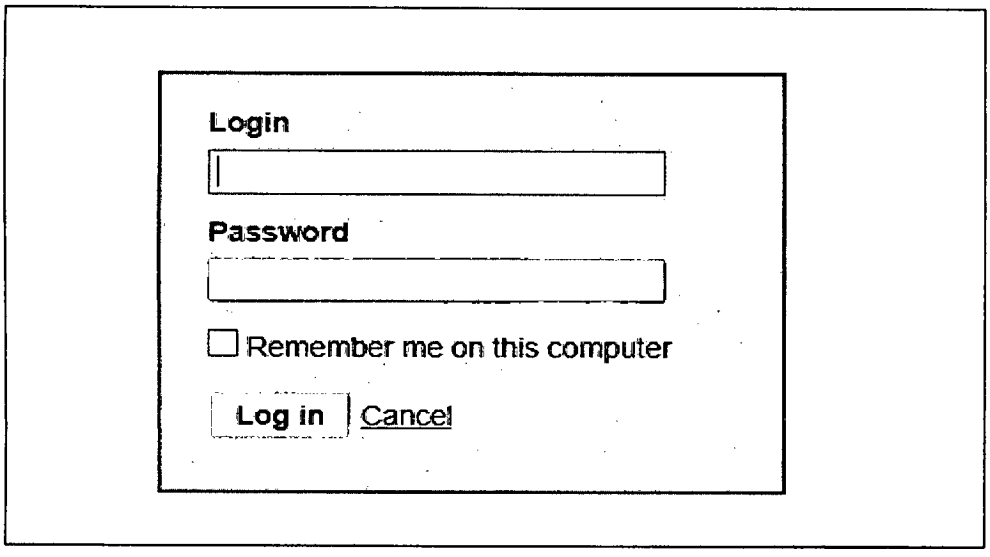


2. ¿Cómo cambiar una regla en el SonarQube?

Para cambiar una regla en el SonarQube primero debes de logearte como administrador del sistema para eso dar clic en la opción "Log in"



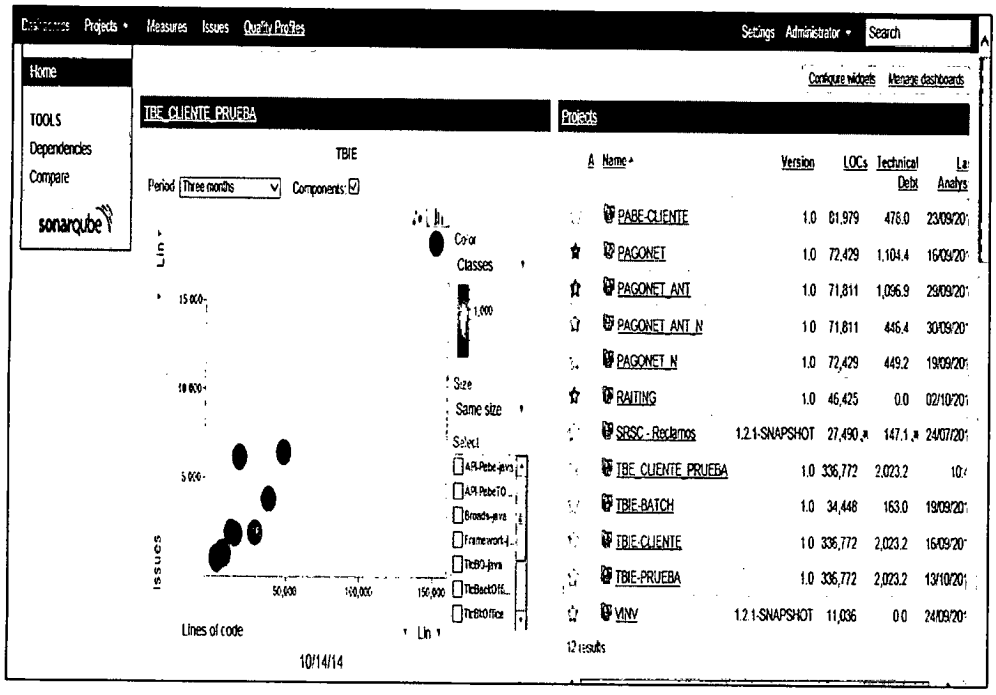
Una vez realizado el paso anterior el sistema solicitara que ingrese su usuario y password.



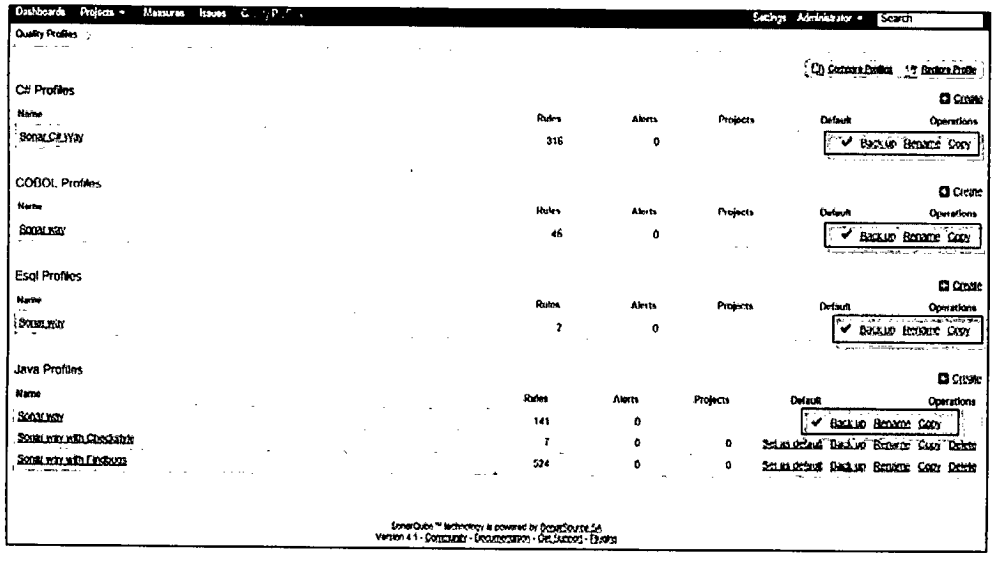
Usuario: admin

(*) Password: admin

Una vez accedido con el perfil de administrador el sistema mostrara la siguiente ventana dar clic en la opción "Quality Profiles"

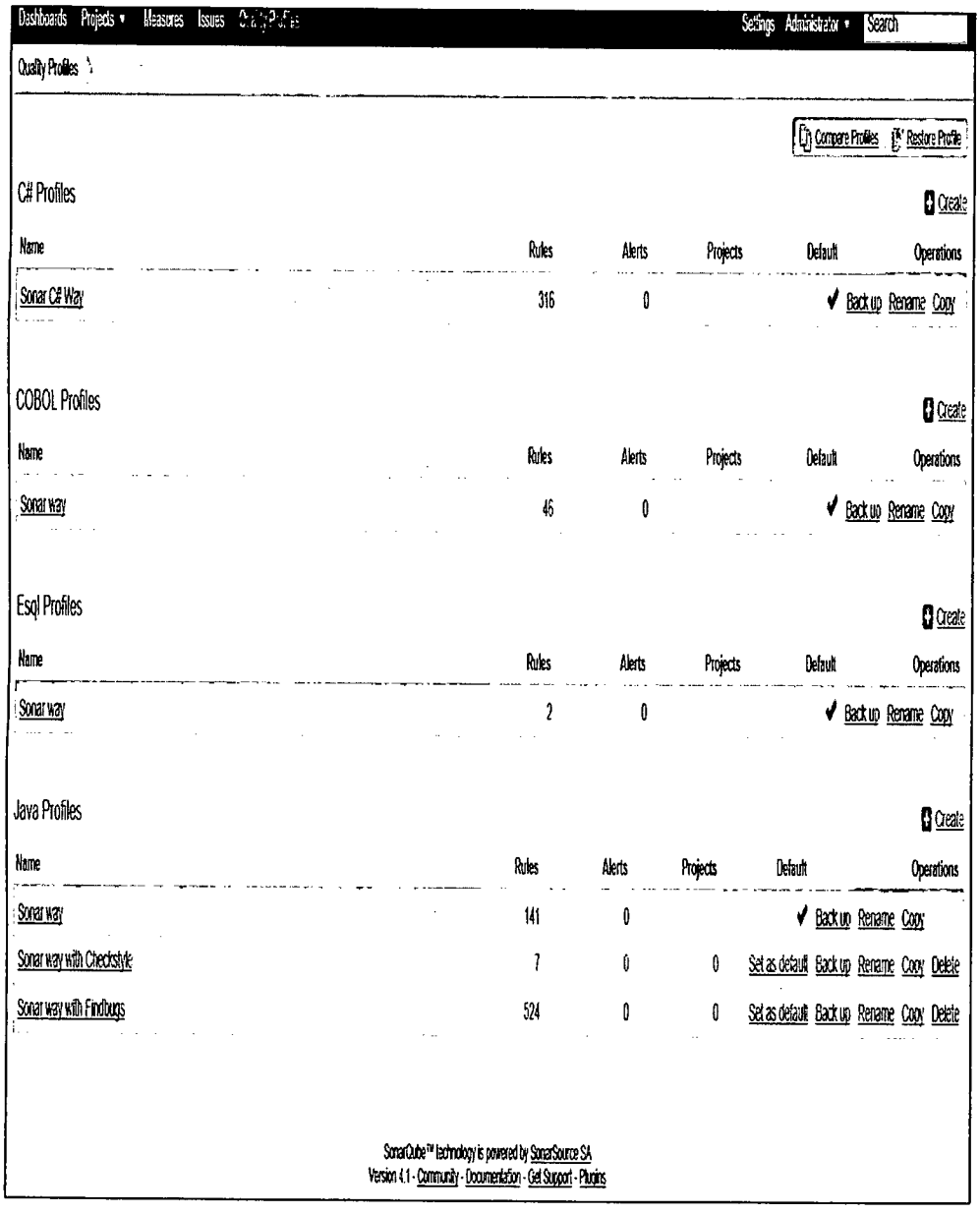


Una vez realizado el paso anterior el sistema mostrara la siguiente ventana donde se observan las opciones a las que se tendrá acceso para crear, editar, seleccionar y eliminar reglas para los diferentes lenguajes de programación(C# Profiles, COBOL Profiles, Esql Profiles, Java Profiles).

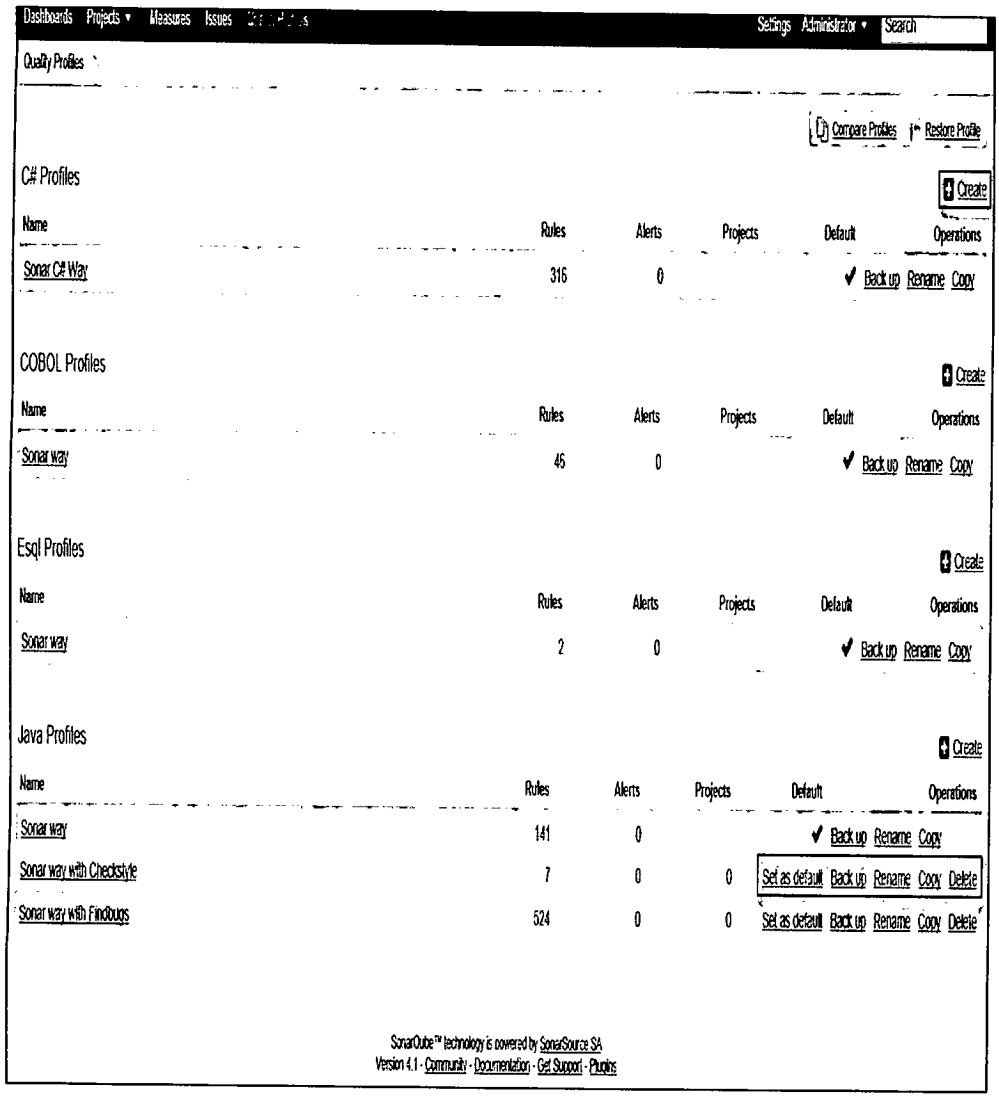


3. ¿Cómo crear una regla?

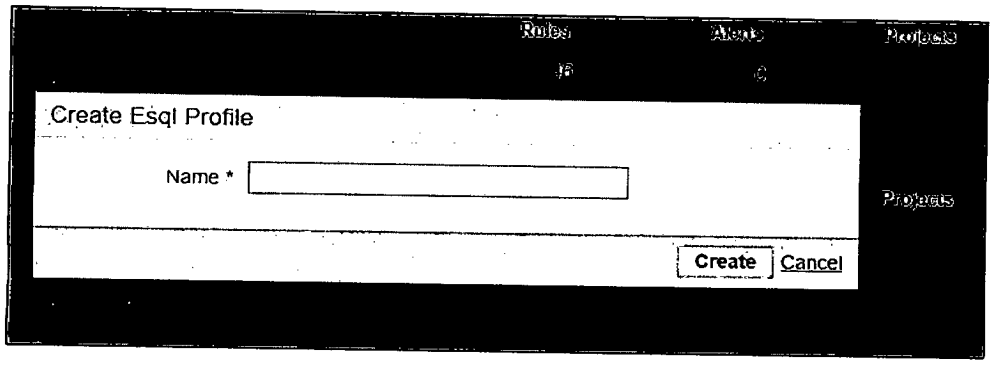
Para crear una regla lo primero que se tiene que hacer es haberse logeado con el usuario administrador (Usuario: admin (*) Password: admin) una vez realizado este paso ingresar a la pestaña "Quality Profiles" donde se abrirá la siguiente ventana:



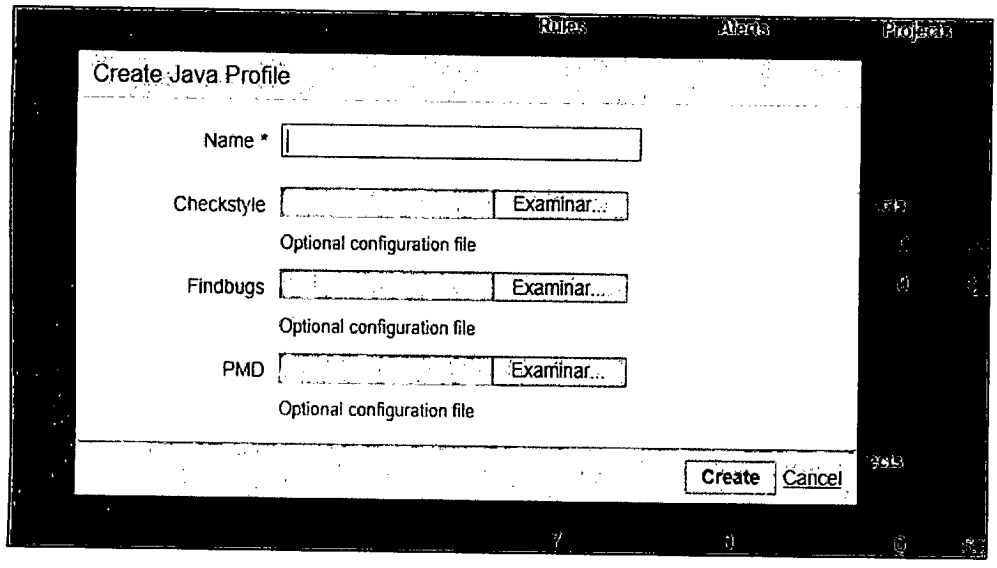
En esta ventana se habilitan las opciones de Set as default, Back up, Create, Rename, Copy y Delete tal como se muestra en la siguiente imagen a continuación:



Para crear una nueva regla dar clic en el botón "Create" (Create) y se abrirá la siguiente ventana modal.



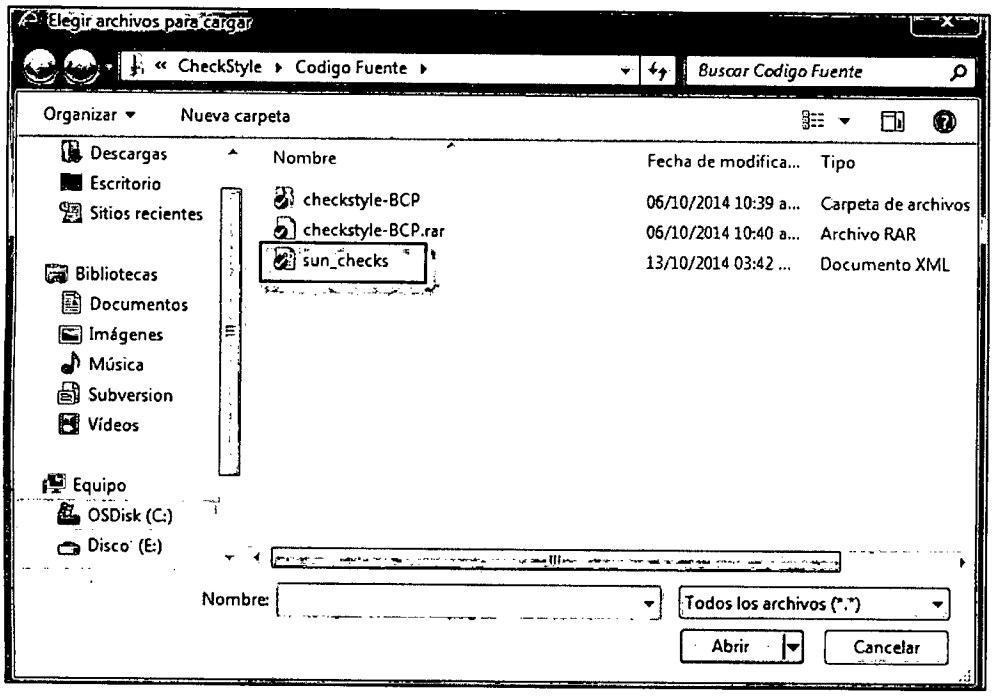
Donde se registrara el nombre el nombre de la regla y seleccionar el Checkstyle dando clic en el botón "Examinar".



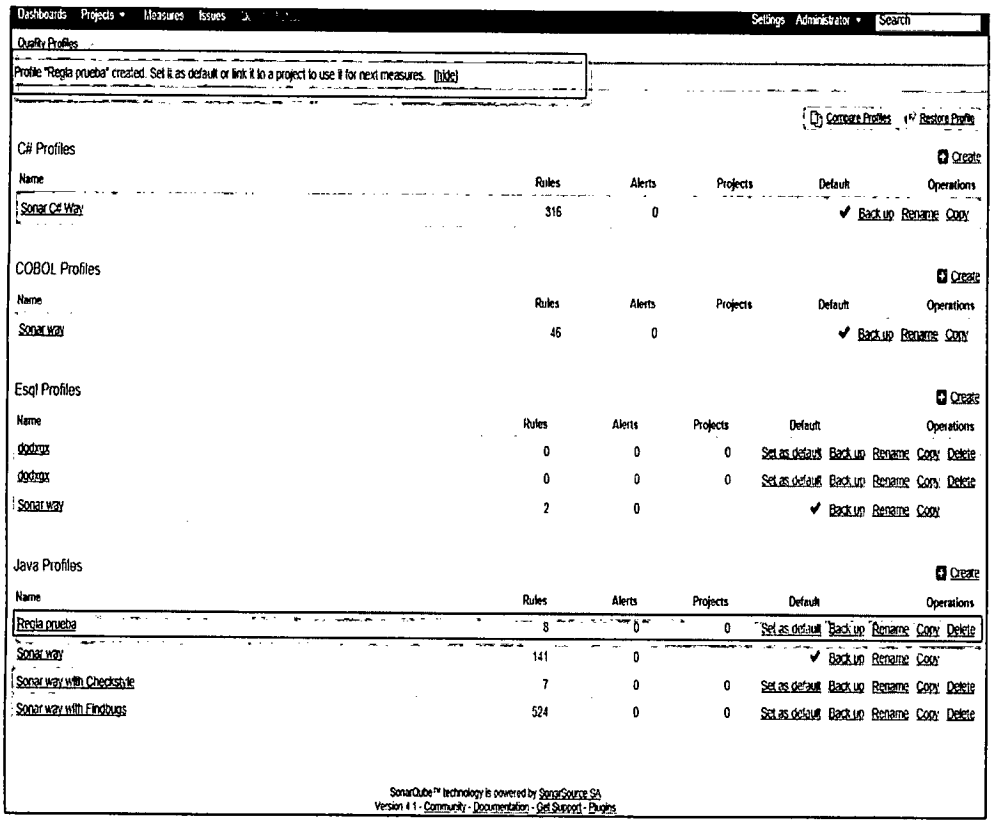
Después de haber dado clic en el botón "Examinar" se abrirá el explorador de objetos de Windows y luego seleccionar la regla generada en la ubicación donde se encuentre. Si se tiene instalado el svn en la pc de escritorio indicar la ruta donde se encuentre y seleccionarla, en el ejemplo se encuentra en la siguiente dirección del svn:

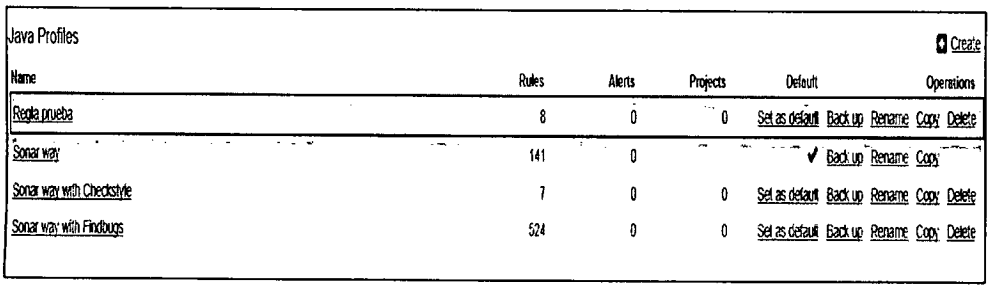
https://7.227.8.11/svn/QC-Everis/01. INICIO/01.02. Herramientas/01.02.01 Validadores de codigo/CheckStyle/Codigo Fuente

Para este ejemplo se seleccionara el archivo sun_checks.xml

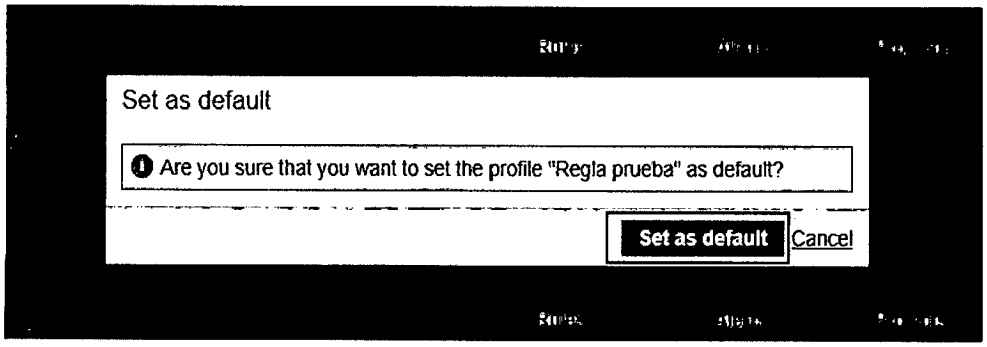


Nota: las reglas generadas deben estar guardadas con extensión ".txt" o extensión ".xml" de lo contrario no serán reconocidas como reglas por el Checkstyle. Una vez seleccionada la regla dar clic en el botón "Create" (Create) para generar la regla, la misma que será visualizada en la siguiente ventana:



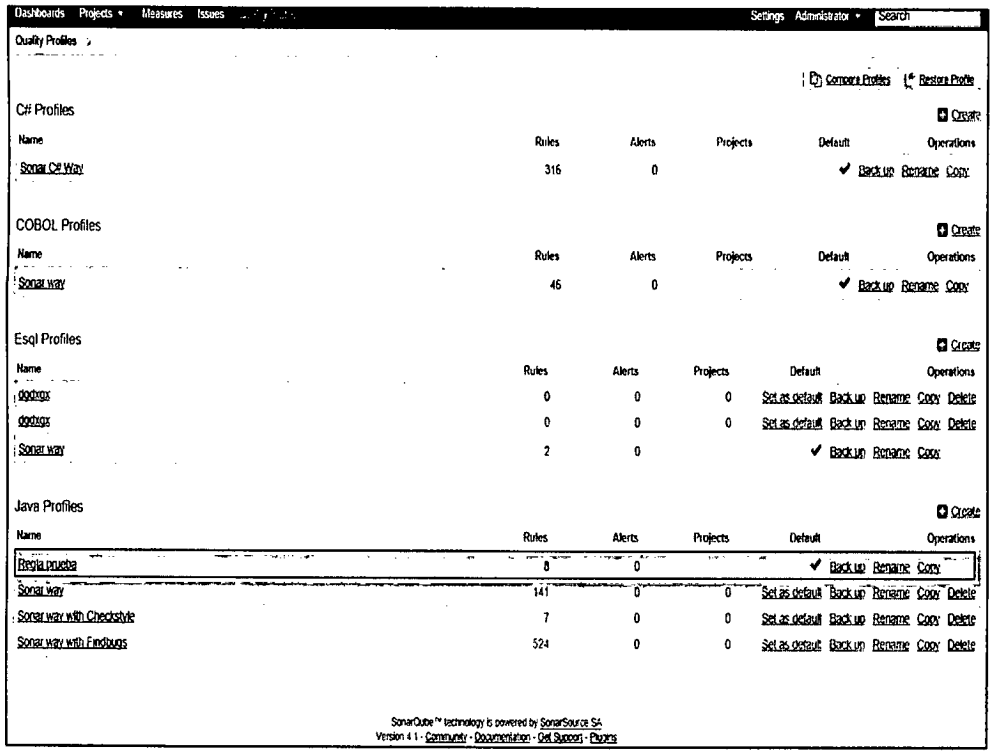


Para seleccionar la regla generada como predeterminada dar clic en el opción "Set as default" (Set as default), una vez dado clic en esta opción se mostrara la siguiente ventana:



Para determinar la regla como predeterminada dar clic en el botón "Set as default"

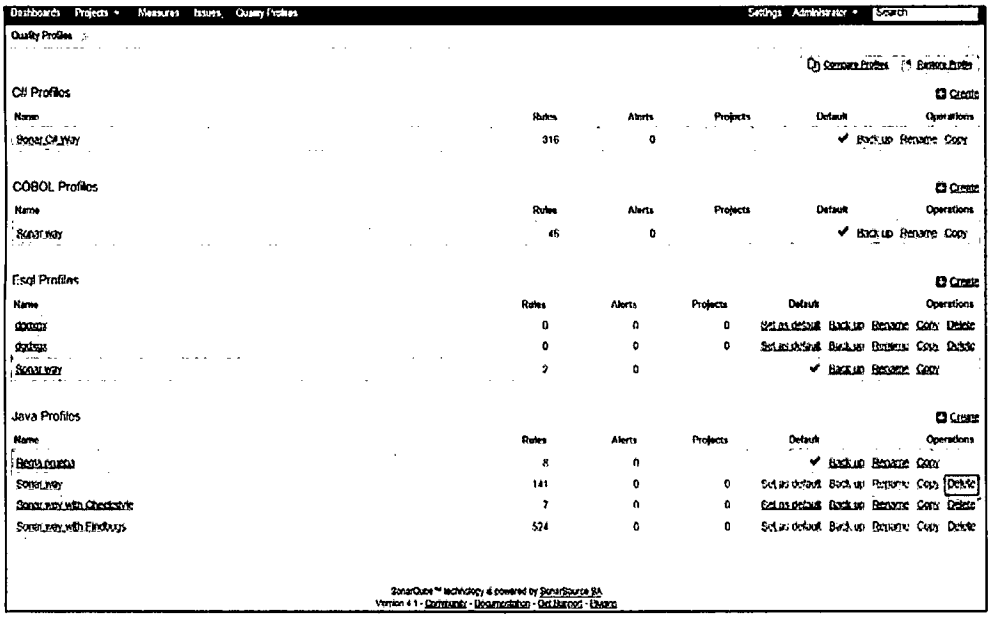
(**Set as default**), A continuación se muestra la ventana "Quality Profiles", en esta ventana se visualizara el icono check (✔) al lado de la regla a predeterminar, esto nos indica que los cambios fueron realizados con éxito.



Nota: no se puede seleccionar como regla predeterminada más de una regla por cada tipo de tecnología(C# Profiles, COBOL Profiles, Esql Profiles, Java Profiles).

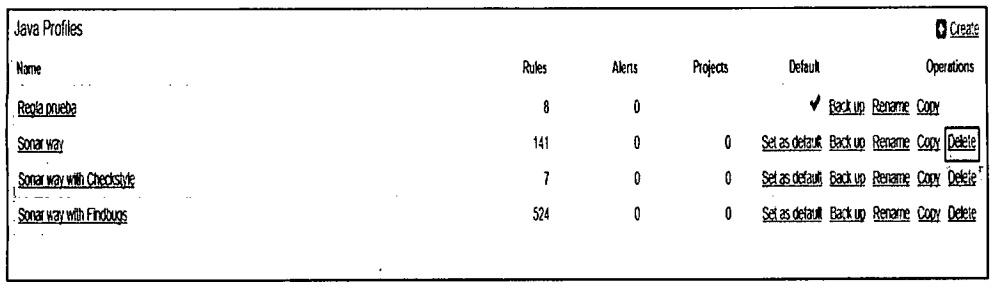
4. ¿Cómo eliminar una regla?

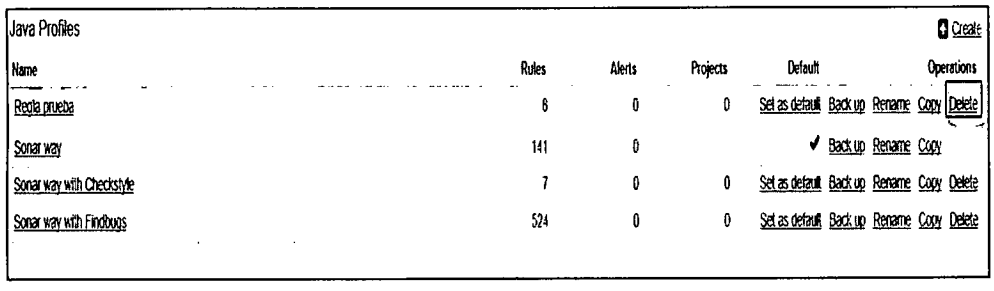
Para eliminar una regla lo primero que se debe hacer es ir a la ventana "Quality Profiles" una vez ingresado se visualizaran una lista de reglas ya sean propias del SonarQube o en algunos casos reglas generadas por el usuario a conveniencia, si desea eliminar una regla lo primero que se deberá realizar es seleccionar la regla a eliminar y dar clic en la opción "Delete" (Delete) tal como se muestra a continuación en la siguiente ventana:



Si esta opción no se encuentra visible, lo primero que deberá realizar es dejar de utilizar la regla como regla predeterminada, acto siguiente se

habilitara la opción "Delete" de no ser este el caso dar clic en la opción delete tal como se muestra a continuación en la siguiente imagen:





Una vez dado clic en la opción "Delete" se visualizara la siguiente ventana popup, para eliminar la regla se deberá dar clic en el botón "Delete" (**Delete**) de lo contrario dar clic en el botón "Cancel"

