

# INFORME DE CALIDAD DE PRODUCTO

Autor	Andrés Barrado Martín
Número de Sprint	Sprint 01
Historia de Usuario	Listar Gasolineras
Fecha	26/10/2016

## **1. Introducción**

En este documento se recoge el análisis de calidad del proyecto en el que se ha implementado la primera historia de usuario de este primer sprint, Listar Gasolineras.

El documento se compone de dos secciones principales, aparte de su correspondiente introducción. En la primera sección se hablará de los resultados obtenidos, la cual se dividirá en cuatro partes, evidencias, medidas, estructura y evolución del proyecto.

La última sección propondrá una serie de mejoras para solventar las evidencias detectadas.

## **2. Resultados del análisis**

A la hora de realizar el análisis de calidad he utilizado el Quality Gate y Quality Profile que SonarQube me ofrece por defecto.

Quality Gate  
(Default) [SonarQube way](#)

Quality Profiles  
(Java) [Sonar way](#)

A continuación, voy a hablar un poco de los resultados obtenidos por el análisis.

Evidencias

En primer lugar, se han encontrado un total de 137 evidencias, las cuales implicarán una deuda técnica de dos días. Sonar clasifica estas evidencias por su severidad, y las evidencias que se han encontrado tras el análisis son de tipo bloqueante (1), crítica (11), mayor (48), menor (77) y de información (0).

Issues 137	Technical Debt	1 Blocker	1
	2d	11 Critical	11
	Reliability Remediation Effort	48 Major	48
	2h 20min	77 Minor	77
	Security Remediation Effort	0 Info	0
	2h		

A continuación, voy a mostrar ejemplos de evidencias de distinta severidad.

Bloqueante:

Remove this return statement from this finally block. [...](#)

hace 7 horas L38 [🔗](#) [🔍](#) [🔧](#)

Bug 🔴 Blocker 🔵 Open Not assigned 30min effort

[cert, cwe, error-handling](#) >

Crítica:

Define and throw a dedicated exception instead of using a generic one. [...](#)

hace 7 horas L20 [🔗](#) [🔍](#) [🔧](#)

Vulnerability 🔴 Critical 🔵 Open Not assigned 20min effort

[cert, cwe, error-handling](#) >

Mayor:

Replace the type specification in this constructor call with the diamond operator ("<>"). (sonar.java.source not set. Assuming 7 or greater.) [xxx](#) hace 7 horas L22   
 Code Smell Major Open Not assigned 1min effort clumsy >

Menor:

Rename this package name to match the regular expression `^[a-z]+(\.[a-z][a-z0-9]*)*$`. [xxx](#) hace 7 horas L1   
 Code Smell Minor Open Not assigned 10min effort convention >

A la hora de mirar si el código cumple determinadas reglas, solo hay que ir a la pestaña rule y elegir la regla que quieres aplicar. En mi caso me interesa ver si algún método tiene demasiada complejidad y para ello sonar me ofrece la regla “Methods should not be too complex”.

<input checked="" type="checkbox"/> Rule	
Package names should comply with a naming convention	21
Public types, methods and fields (API) should be documented	18
String literals should not be duplicated	14
String literals should be placed on the left	13
Sections of code should not be "commented out"	9
Exception handlers should preserve the original exception	7
Local variable and method parameter names should be meaningful	6
Multiple variables should not be declared on the same line	5
Source files should not have any duplicated code	4
Method names should comply with a naming convention	4
Generic exceptions should never be thrown	4
@FunctionalInterface annotation should be used	4
The diamond operator ("<>") should be used	4
Field names should comply with a naming convention	3
Methods should not be too complex	2

A continuación, se puede ver un ejemplo de un método que debería modificarse al tener una alta complejidad ciclomática.

```
@Override
public int compareTo(Gasolinera another) {

//Si el atributo orderByGasoleoA está a true se ordena por precio del diesel
if(orderByGasoleoA){
    if(this.getGasoleo_a())>another.getGasoleo_a())
        return 1;
    else if(this.getGasoleo_a())<another.getGasoleo_a())
        return -1;
    else
        return 0;
}
//En caso contrario se ordena por precio de La gasolina
}else{
    if(this.getGasolina_95())>another.getGasolina_95())
        return 1;
    else if(this.getGasolina_95())<another.getGasolina_95())
        return -1;
    else
        return 0;
}
}

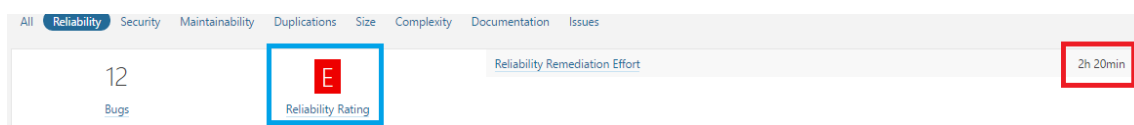
}

//compareTo
}
```

File	
om/nodry/nodry/Utils/ParserJSONTest.java	31
va/com/nodry/nodry/Utils/ParserJSON.java	26
va/com/nodry/nodry/Datos/Gasolinera.java	17
Presentacion/GasolinerasArrayAdapter.java	9
y/com/nodry/nodry/Utils/RemoteFetch.java	6
nodry/nodry/Datos/GasolinerasDAOTest.java	5
y/nodry/Presentacion/IntegrationTest1.java	5
y/nodry/Presentacion/IntegrationTest3.java	5
m/nodry/nodry/Utils/RemoteFetchTest.java	5
n/nodry/nodry/Datos/GasolinerasDAO.java	5
nodry/Presentacion/GetGasolinerasTask.java	4
nodry/nodry/Presentacion/CancelarTask.java	3
nodry/Negocio/GestionGasolinerasTest.java	3
y/nodry/nodry/Datos/IGasolinerasDAO.java	2
ry/nodry/Negocio/IGestionGasolineras.java	2

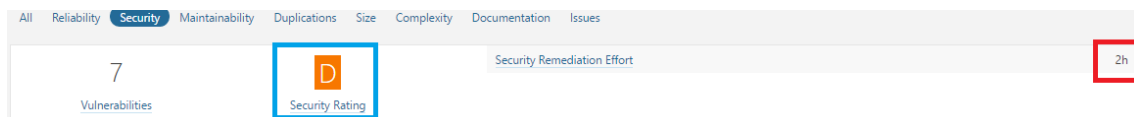
En las imágenes que podéis ver a continuación, he marcado en azul la calificación SQALE y en rojo la deuda técnica.

En primer lugar, la medida de confiabilidad referente a la calificación SQALE es de una E, la más baja dentro de la calificación SQALE. Esto puede provocar que el usuario quiera realizar una tarea, y el sistema falle. Al ver los diferentes archivos, la clase GasolineraDAO es la única que tiene una calificación de E, por lo que, si se mejorara esta clase, seguramente se obtendría una diferente calificación.

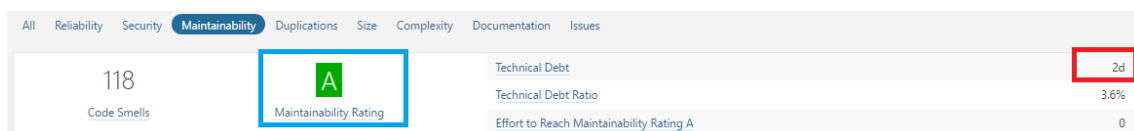


La medida seguridad tiene una calificación SQALE de D, mejor que la calificación obtenida en la medida confiabilidad, pero sigue siendo una medida bastante mala, a pesar de esto podría remediarse fácilmente con dos horas de esfuerzo.

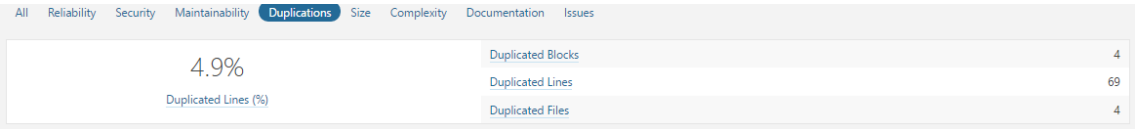
La clase ParserJSONTest es la que más bugs y vulnerabilidades tiene, por lo que convendría mejorarla.



La medida mantenibilidad tiene la mejor calificación SQALE posible, tiene una deuda técnica de dos días porque tiene muchas evidencias, pero es un código mantenible.

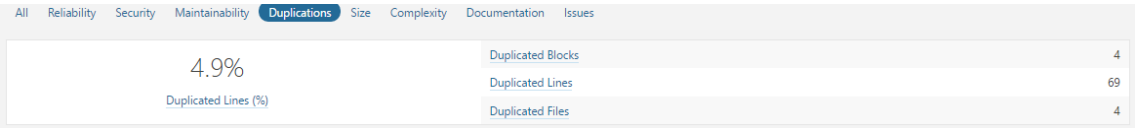


En la imagen que muestro a continuación, podéis ver información sobre el número de bloques, líneas y archivos duplicados.



Estructura

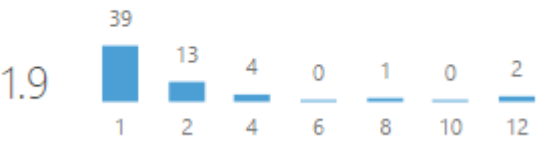
A continuación, se puede ver una imagen que muestra el número total de líneas de código (889), las cuales se pueden clasificar por líneas (1403), statements (248), funciones (65), clases (21), ficheros (21) y directorios (8).



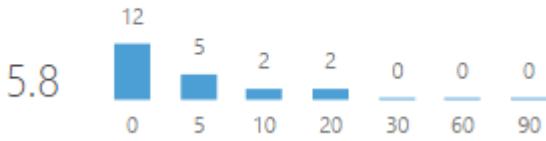
A continuación, se ofrecen tres imágenes las cuales nos muestran los valores de complejidad total, por fichero y función.



Complexity / Function



Complexity / File



Tras ver estos gráficos, considero que los ficheros tienen una complejidad media alta pero dentro de un intervalo razonable.

main/java/com/nodry/nodry/Datos/Gasolinera.java	27
main/java/com/nodry/nodry/Utils/ParserJSON.java	25
androidTest/java/com/nodry/nodry/Utils/ParserJSONTest.java	14
main/java/com/nodry/nodry/Presentacion/GetGasolinerasTask.java	10

En la imagen anterior, se ven los ficheros que presentan una mayor complejidad.

El código está comentado en un 17,9 %, es un porcentaje bastante pequeño y podría provocar pérdidas de tiempo para posteriores implementaciones.

All	Reliability	Security	Maintainability	Duplications	Size	Complexity	Documentation	Issues
<div>17.9%</div> <div>Comments (%)</div>							Comment Lines	194
							Public API	68
							Public Documented API (%)	72.1%
							Public Undocumented API	19

Como se puede ver a continuación, hay ficheros que tienen APIs públicas sin documentar.

Public Undocumented API

19

List	Tree	History
androidTest/java/com/nodry/nodry/Utils/ParserJSONTest.java		4
main/java/com/nodry/nodry/Utils/ParserJSON.java		4
androidTest/java/com/nodry/nodry/Presentacion/IntegrationTest3.java		3
androidTest/java/com/nodry/nodry/Presentacion/IntegrationTest1.java		3
androidTest/java/com/nodry/nodry/Utils/RemoteFetchTest.java		2
androidTest/java/com/nodry/nodry/Datos/GasolinerasDAOTest.java		2
test/java/com/nodry/nodry/Negocio/GestionGasolinerasTest.java		1

## Evolución del proyecto

Este informe de calidad es de la primera historia de usuario por lo que no se pueden ver estos aspectos.

### 3. Acciones correctivas

1. Como se mostró anteriormente, tenemos muy mal los parámetros de confiabilidad y seguridad.

Para mejorar la medida confiabilidad, se deberían mejorar las clases GasolinerasDAO y ParserJSONTest, las cuales tienen unos niveles de confiabilidad muy bajos y necesitan un mayor número de horas de trabajo.






List	Tree	History
androidTest/java/com/nodry/nodry/Utils/ParserJSONTest.java		40min
main/java/com/nodry/nodry/Datos/GasolinerasDAO.java		40min

Y para mejorar la medida seguridad, se deberían mejorar las clases ParserJSONTest y RemoteFetchTest, las cuales tienen unos niveles de seguridad muy bajos y necesitan un mayor número de horas de trabajo.

androidTest/java/com/nodry/nodry/Utils/ParserJSONTest.java		30min
androidTest/java/com/nodry/nodry/Utils/RemoteFetchTest.java		20min

2. Para mejorar el código y conseguir así que fuese más claro, deberíamos aumentar el porcentaje de documentación ya que el valor 17.9% es un valor extremadamente pequeño.

3. Por supuesto, prioritariamente deberíamos solucionar la evidencia bloqueante que nos indica tener y las evidencias críticas cuyo número asciende a 11. A continuación sería muy conveniente solucionar las evidencias mayor.

 Blocker	<a href="#">1</a>
 Critical	<a href="#">11</a>
 Major	<a href="#">48</a>
 Minor	<a href="#">77</a>
 Info	<a href="#">0</a>