

# INFORME DE CALIDAD DE PRODUCTO

Autor	Jorge Pereda
Número de Sprint	Sprint 001
Historia de Usuario	Listar Gasolineras
Fecha	24/10/2016

## Introducción.

El objetivo de la actividad es realizar un análisis de calidad interna de producto software mediante la herramienta SonarQube.

Este análisis será aplicado al proyecto “AppGasolineras”, sobre la historia de usuario “Listar Gasolineras”, en el cual se aplicarán las medidas convenientes para mejorar y/o maximizar la calidad del proyecto.

## Análisis

Para el primer análisis se procederá con la configuración por defecto de SonarQube en cuanto a Quality Gate y Quality Profile.

Quality Gate:

Metric	Over Leak Period	Operator	Warning	Error
Coverage on New Code	Always	is less than	<input type="text"/>	<input type="text" value="80"/>
New Bugs	Always	is greater than	<input type="text"/>	<input type="text" value="0"/>
New Vulnerabilities	Always	is greater than	<input type="text"/>	<input type="text" value="0"/>
Technical Debt Ratio on New Code	Always	is greater than	<input type="text"/>	<input type="text" value="5"/>

## Quality Profile:

Quality Profiles / Java

Sonar way

Rules	Active	Inactive
Total	268	111
Bugs	89	13
Vulnerabilities	18	13
Code Smells	161	85

Para evitar fallos no controlados sobre clases autogeneradas del proyecto, la ruta sobre la que se aplicará el escaner será solo en el contenido de la carpeta “src” del proyecto.

Fijándonos primero en la calificación SQALE, daremos prioridad a las medidas cuya calificación sea D o peor, ya que eso quiere decir que estamos ante un ratio de deuda técnica superior al 50%.

Una vez nos hemos fijado en la calificación SQALE, y es C o mejor, nos centraremos en otros aspectos.

Los aspectos a los cuales se les darán más importancia a la hora de analizar serán los bugs y vulnerabilidades con una severidad “Blocker”, “Critical” o “Major”, en ese orden de importancia.

Después procederemos con los “Code Smell”, dando el mismo orden de importancia que con los bugs según su severidad.

En caso de existir duplicaciones de código, se revisarán a continuación.

Una vez revisados estos apartados, si el proyecto dispone de tiempo sobrante se realizarán las correcciones a la complejidad del código, a la documentación y a los bugs, vulnerabilidades y “Code Smell” poco severos en ese orden.

## Resultados del análisis

Quality Gate

Passed

El proyecto ha pasado el Quality Gate, pero en este momento, al ser la primera versión, no es algo que debamos tener muy en cuenta, ya que el Quality Gate aplicado se centra sobretudo en los nuevos fallos respecto a una versión anterior.


En cuanto a Bugs y Vulnerabilidades, pese a que parecen pocos, la calificación SQALE es muy mala, por lo que deberemos centrarnos en mejorarlo.

Si nos fijamos en el esfuerzo necesario, vemos que además podemos solucionarlos rapidamente.

Reliability			
3		Reliability Remediation Effort	30min
Bugs	Reliability Rating		
Security			
3		Security Remediation Effort	50min
Vulnerabilities	Security Rating		

Las medidas de mantenibilidad son muy buenas, recibiendo una calificación A.

Aún así, debido a que son muchos Code Smells, la deuda técnica sube a 1 día, pero no llega a ser un 5%, por lo tanto será algo secundario a la hora de mejorarlo.

Maintainability			
79		Technical Debt	1d 1h
Code Smells	Maintainability Rating	Technical Debt Ratio	4.9%
		Effort to Reach Maintainability Rating A	0

El código del proyecto está completamente libre de duplicaciones, no requiere medidas correctoras.

### Duplications

	0.0%	0
Duplications		Duplicated Blocks

Fijándonos en el tamaño del proyecto, vemos que el número de ficheros y de clases es muy parecido, de forma que no estamos sobrecargando cada fichero.

Size													
378 Lines of Code	<table> <tr><td>Lines</td><td>625</td></tr> <tr><td>Statements</td><td>131</td></tr> <tr><td>Functions</td><td>45</td></tr> <tr><td>Classes</td><td>14</td></tr> <tr><td>Files</td><td>11</td></tr> <tr><td>Directories</td><td>6</td></tr> </table>	Lines	625	Statements	131	Functions	45	Classes	14	Files	11	Directories	6
Lines	625												
Statements	131												
Functions	45												
Classes	14												
Files	11												
Directories	6												

Complexity							
73 Complexity	<table> <tr><td>Complexity / Function</td><td>1.6</td></tr> <tr><td>Complexity / File</td><td>6.6</td></tr> <tr><td>Complexity / Class</td><td>5.2</td></tr> </table>	Complexity / Function	1.6	Complexity / File	6.6	Complexity / Class	5.2
Complexity / Function	1.6						
Complexity / File	6.6						
Complexity / Class	5.2						

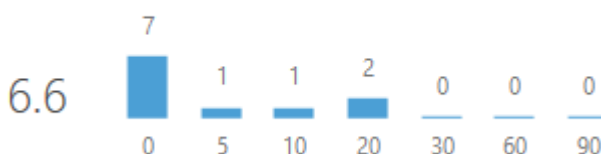
En cuanto a la complejidad de los métodos, podemos decir que la calidad del código fuente es buena, ya que la mayoría tienen una complejidad muy simple y solo hay uno que suba un poco, pero no de forma excesiva.

#### Complexity / Function



Si nos fijamos en la complejidad de los ficheros si que aumenta un poco, ya que hay dos ficheros con una complejidad superior, que son los que desvian la media. Pero son dos ficheros importantes que no podemos modificar demasiado sin entorpecer el proyecto, estos ficheros son la clase del objeto principal que manejamos, "Gasolinera", y la clase que maneja los datos recibidos del servicio web.

#### Complexity / File



La complejidad en torno a las clases es practicamente igual que la de los ficheros, ya que la relación clases por fichero es casi 1 a 1.

La documentación es muy baja, solo un 13% del proyecto esta comentado. No es algo prioritario, pero convendría aumentar la cantidad de documentación, sobretodo en las clases cuya complejidad hemos visto que es mayor que las demás.

Documentation	
13.5%	Comment Lines 59
Comments (%)	Public API 45
	Public Documented API (%) 26.7%
	Public Undocumented API 33

## Plan de mejora

Para mejorar el proyecto software se seguirán los siguientes pasos, teniendo en cuenta la gestión previa al análisis realizada al principio del documento y observando los resultados obtenidos:

1. Se procederá a corregir los bugs y vulnerabilidades que han resultado en una calificación SQALE D o peor.

Bugs:

En clase “/appgasolineras/Datos/Gasolinera.java”

Override "equals(Object obj)" to comply with the contract of the "compareTo(T o)" method. ...  
Bug ▼ ⬆ Critical ▼ ○ Open ▼ Not assigned ▼ 15min effort Comment

Vulnerabilidades:

En clase “appgasolineras/Datos/GasolineraDAO.java”

Use a logger to log this exception. ...  
Vulnerability ▼ ⬆ Critical ▼ ○ Open ▼ Not assigned ▼ 10min effort Comment

Define and throw a dedicated exception instead of using a generic one. ...  
Vulnerability ▼ ⬆ Critical ▼ ○ Open ▼ Not assigned ▼ 20min effort Comment

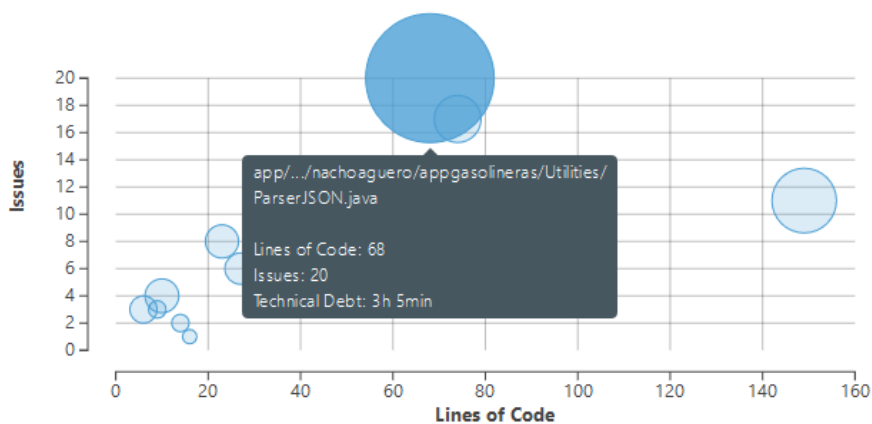
En clase “appgasolineras/ExampleUnitTest.java”

Define and throw a dedicated exception instead of using a generic one. ...  
Vulnerability ⬇ 🔴 Critical ⬇ 🔵 Open ⬇ Not assigned ⬇ 20min effort 💬 Comment

2. Se documentará de forma más completa las clases con alta complejidad.

app/src/main/java/com/example/nachoaguero/appgasolineras/Presentacion/ListaGasolinerasActivity.java	26.0
app/src/main/java/com/example/nachoaguero/appgasolineras/Utilities/ParserJSON.java	22.0
app/src/main/java/com/example/nachoaguero/appgasolineras/Datos/Gasolinera.java	21.0

3. Se intentará reducir la deuda técnica atendiendo a mayor severidad de los fallos y después a mayor número de incidencias por fichero, como en el ejemplo de la imagen.



4. Se reducirá en lo posible la deuda técnica producida por el resto de “Code Smells”.

Technical Debt	1d 1h
Added Technical Debt	0
Technical Debt Ratio	4.3%
Technical Debt Ratio on New Code	0.0%
Effort to Reach Maintainability Rating A	0

## Evolución del proyecto

Los bugs han conseguido reducirse de 3 a 1, aunque la calificación SQALE se mantiene en D debido a que el bug restante sigue siendo crítico.

