

# INFORME DE CALIDAD DE PRODUCTO

Autor	Alba Zubizarreta Solá
Número de Sprint	Sprint 2
Historia de Usuario	US-169386 Mostrar detalle gasolinera
Fecha	09/11/2016

Se realiza un análisis estático de código con la herramienta *SonarQube*. El código analizado corresponde con las implementaciones realizadas en el *primer y segundo sprint*.

*NOTA:* Las comparaciones de SonarQube con versiones anteriores no son válidas, ya que solo contemplan un par de errores sencillos de la propia historia de usuario que fueron solucionados y no la evolución de la calidad del código respecto al informe anterior.

## 1. Resultados del análisis

La ejecución del análisis se ha llevado a cabo con la configuración por defecto de *SonarQube*. Dentro de esta configuración por defecto se está aplicando la *QualityProfile Sonar way*, la cual consta de 268 reglas de Java. En cuanto al *QualityGate* utilizado, se ejecuta *SonarQube Way*, por lo que se aplican las métricas *Coverage on New Code*, *New Bugs*, *New Vulnerabilities* y *Technical Debt Ratio on New Code*.

La ejecución del análisis nos deja una serie de métricas que nos indican el estado en el que se encuentra el proyecto. Como se puede observar en la *Figura 1*, tenemos un total de 226 evidencias sin resolver. Sin embargo, el estado del proyecto no es malo ya que por la estimación en esfuerzo vemos que deberíamos invertir una hora y media para arreglar los bugs y vulnerabilidades.

Issues				Effort			
Type				Type			
Bug			7	Bug			1h
Vulnerability			2	Vulnerability			30min
Code Smell			217	Code Smell			2d
Resolution				Resolution			
Unresolved	226	Fixed	0	Unresolved	3d	Fixed	0
False Positive	0	Won't fix	0	False Positive	0	Won't fix	0
Removed	0			Removed	0		
Severity				Severity			
Blocker	0	Minor	151	Blocker	0	Minor	1d
Critical	3	Info	1	Critical	45min	Info	0
Major	71			Major	1d		

Figura 1. Número de vulnerabilidades y esfuerzo asociado.

Nos encontramos con que no tenemos errores bloqueantes, pero sí 3 críticos, mayores y menores. En la Figura 2 se muestran ejemplos de cada uno de estos tipos de errores.

<input type="checkbox"/>	Use a logger to log this exception. ...	hace 15 minutos	L61	error-handling	>
	Vulnerability	Critical	Open	Not assigned	10min effort
<input type="checkbox"/>	Either log or rethrow this exception. ...	hace 15 minutos	L60	cert, error-handling, suspicious	>
	Bug	Major	Open	Not assigned	10min effort
<input type="checkbox"/>	Introduce a new variable instead of reusing the parameter "bForceLocal". ...	hace 15 minutos	L47	misra, pitfall	>
	Bug	Minor	Open	Not assigned	5min effort

Figura 2. Ejemplo de error crítico, mayor y menor.

Si filtramos para obtener las issues relacionadas con malas prácticas de programación, por ejemplo, ya que podemos filtrar por muchos tipos, nos aparecen errores menores y mayores.



Figura 3. Ejemplo de issues encontradas tras filtrar por el tag “bad-practice”

Los errores que nos aparecen podrían intuirse como poco importantes a pesar de ser mayor porque son relacionados con las buenas prácticas, pero por supuesto es conveniente resolverlos para tener un código más claro y legible.<sup>4</sup>

Si buscamos incidencias según la regla que las detecta, por ejemplo, la regla “architectural constraints” que salvaguarda los accesos directos de presentación a persistencia en arquitecturas de capas no encontramos ningún error. En un primer lugar esto se debe a que la regla no está activa en la configuración por defecto de Sonar. Procedemos a activar esta regla en nuestro QualityProfile y observamos que tampoco nos detecta ningún issue, por lo que estamos respetando las restricciones en la arquitectura de 3 capas.

Si se desea también se pueden observar las incidencias de cada fichero por separado. Por ejemplo, vamos a aplicarlo al fichero *GestionGasolineras.java*. Al seleccionarla vemos que tiene 5 errores como se muestra en la siguiente figura (Figura 4).

Proyecto NoDry main/java/com/nodry/nodry/Negocio/GestionGasolineras.java

<input type="checkbox"/>	Rename this package name to match the regular expression <code>^[a-z]+(\.[a-z][a-z0-9]*)*\$</code> . <small>...</small>	hace una hora	L1	\$	convention	>
<input type="checkbox"/>	Code Smell <span>Minor</span> <span>Open</span> <span>Not assigned</span> 10min effort <a href="#">Comment</a>					
<input type="checkbox"/>	Rename this local variable name to match the regular expression <code>^[a-z][a-zA-Z0-9]*\$</code> . <small>...</small>	hace una hora	L28	\$	convention	>
<input type="checkbox"/>	Code Smell <span>Minor</span> <span>Open</span> <span>Not assigned</span> 2min effort <a href="#">Comment</a>					
<input type="checkbox"/>	Rename this local variable name to match the regular expression <code>^[a-z][a-zA-Z0-9]*\$</code> . <small>...</small>	hace una hora	L35	\$	convention	>
<input type="checkbox"/>	Code Smell <span>Minor</span> <span>Open</span> <span>Not assigned</span> 2min effort <a href="#">Comment</a>					
<input type="checkbox"/>	The Cyclomatic Complexity of this method "getMasBaratas" is 19 which is greater than 10 authorized. <small>...</small>	hace una hora	L47	\$	brain-overload	>
<input type="checkbox"/>	Code Smell <span>Major</span> <span>Open</span> <span>Not assigned</span> 19min effort <a href="#">Comment</a>					
<input type="checkbox"/>	Document this public method. <small>...</small>	hace una hora	L47	\$	convention	>
<input type="checkbox"/>	Code Smell <span>Minor</span> <span>Open</span> <span>Not assigned</span> 10min effort <a href="#">Comment</a>					

Figura 4. Issues encontradas en el fichero GestionGasolineras.java

Pasamos ahora a analizar las medidas de confiabilidad, seguridad y mantenibilidad del proyecto.

Para empezar vamos a analizar la confiabilidad del proyecto, que como se puede observar en la Figura 5, aparecen 7 bugs, los cuales introducen un total de 1 hora y 15 minutos de deuda técnica en el sistema. Estos errores de confiabilidad nos hacen descender la clasificación SQALE de la aplicación hasta una D, ya que entre estos 6 errores se encuentra 1 crítico. El archivo que contiene un mayor número de errores de confiabilidad es *Utils.java* con un total de 2 bugs como apreciamos en la Figura 6.

Estos errores hacen que la clasificación SQALE del proyecto sea una D, porque hay una de las clases que toma esta calificación D que es *GetGasolinerasTask.java* con un total de 12 issues.

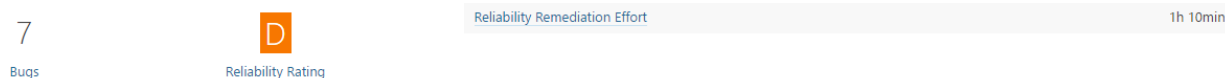


Figura 5. Resultado del análisis de confiabilidad del proyecto.

Al ver las medidas de seguridad de la aplicación, nos encontramos con que tan solo tenemos 2 vulnerabilidades (Figura 6). A pesar de esto, obtenemos una clasificación SQALE de D, ya que una de estas dos incidencias es crítica. Estos 2 fallos de seguridad aumentan la deuda técnica de la aplicación en 30 minutos, siendo el archivo *DataFetch.java* el que contiene la vulnerabilidad crítica y que introduce mayor deuda técnica (Figura 7), al igual que nos ocurría en la versión anterior.


2		Security Remediation Effort	30min
<a href="#">Vulnerabilities</a>	<a href="#">Security Rating</a>		

Figura 6. Resultado del análisis de seguridad del proyecto.

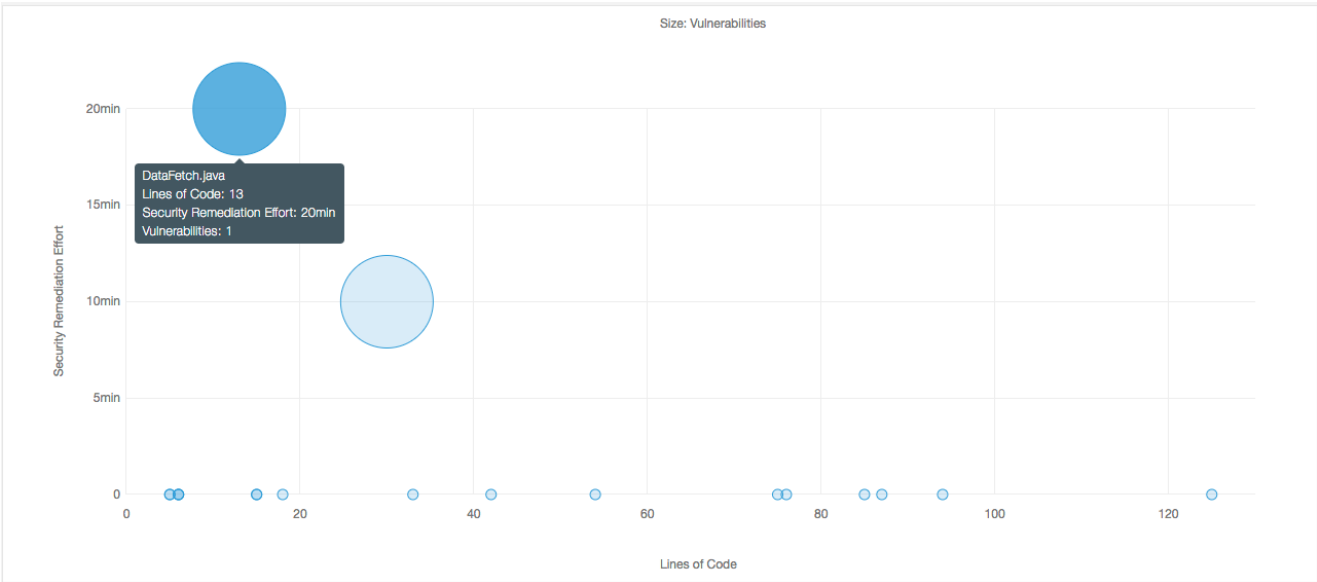


Figura 7. Bubble chart del análisis de seguridad del proyecto.

Analizando la mantenibilidad de la aplicación, obtenemos un buen resultado con una calificación SQALE de A. Vemos que en el proyecto tenemos 217 Code Smells que introducen una deuda técnica de unos 2 días y 7 horas, pero el radio de deuda técnica no supera el 4.2%, justificando así nuestra buena puntuación.

Considero que no hay que invertir tiempo en este aspecto ya que tenemos una buena puntuación así como en otros aspectos se nos va la calificación SQALE a la letra D.


217		Technical Debt	2d 7h
<a href="#">Code Smells</a>	<a href="#">Maintainability Rating</a>	<a href="#">Technical Debt Ratio</a>	4.2%
		<a href="#">Effort to Reach Maintainability Rating A</a>	0

Figura 9. Resultado del análisis de mantenibilidad del proyecto.

Continuando con el análisis vemos que la cantidad de código duplicado dentro del proyecto es de un 1.8%, teniendo 2 bloques duplicados, 34 líneas duplicadas y 2 ficheros duplicados (Figura 10).

1.8%	<a href="#">Duplicated Blocks</a>	2
<a href="#">Duplicated Lines (%)</a>	<a href="#">Duplicated Lines</a>	34
	<a href="#">Duplicated Files</a>	2

Figura 10. Análisis del código duplicado del proyecto.

A continuación vamos a analizar la estructura del proyecto.

En este punto el proyecto consta de un total de 1897 líneas de código, divididas en 4 directorios que contienen los 21 ficheros java. Utilizamos un total de 103 funciones y 511 declaraciones de variables y atributos. En total el proyecto tiene una longitud de 1124 líneas (Figura 11).



Figura 11. Tamaño del proyecto.

En cuanto a la complejidad que nos presenta la aplicación, tenemos una complejidad total en la aplicación de 203. También puede verse esta clasificación por función, obteniendo una media de 2.0, por fichero obteniendo una media de 9.7 o por clase obteniendo una media de 9.7 (Figura 12).



Figura 12. Complejidad del proyecto.

Ahora vamos a ver la clasificación por funciones, ficheros y clases detallada. Vemos como en la clasificación por función hay 68 funciones que tienen complejidad 1, que es la menor, y 13 funciones tienen la mayor complejidad, que es 12. En la clasificación por fichero vemos como 8 ficheros tienen complejidad 0 y 1 fichero tiene una complejidad de 30. (Figura 13).

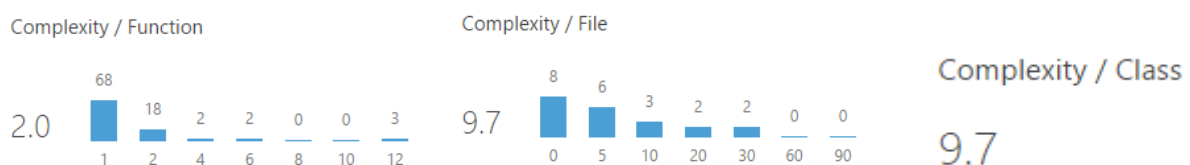


Figura 13. Complejidad por función, fichero y clase.

En el análisis de la documentación observamos que el 20.4% del proyecto son comentarios y que un 74.7% de las librerías públicas están comentadas (Figura 14).

Considero que estos porcentajes están dentro de un rango razonable.

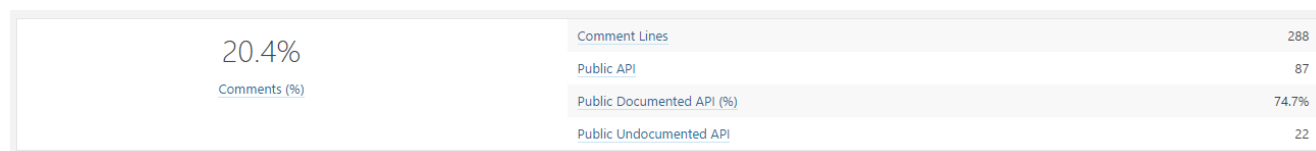


Figura 14. Análisis de los comentarios del proyecto.

Para analizar la evolución del proyecto, vamos a comparar los datos con el análisis realizado en la anterior historia de usuario, que ha sido implementada en este mismo sprint.

La comparación la realizo con los datos obtenidos por mi compañero de grupo *Orlando Britto Herreros*.

En el análisis de la historia de usuario anterior teníamos 152 issues mientras que en este nuevo análisis vemos que el número se ha incrementado hasta 226 issues. Este dato no es bueno para el desarrollo del proyecto pero está justificado por muchas nuevas funciones que se han introducido. Además, al incrementarse el número de issues vemos que lógicamente se ha incrementado la deuda técnica pasando de aproximadamente 2 días a 3 días, lo que tampoco nos dice nada bueno acerca del proyecto.

La calificación SQALE del proyecto se mantiene en una D en seguridad a pesar de haber aumentado en 1 el número de bugs. El motivo sigue siendo el anteriormente mencionado, que al añadir funcionalidades extra han aparecido muchos errores a solucionar.

Mantenemos la A de la calificación en mantenibilidad a pesar de aumentar el número de code smells de 145 a 216.

El tamaño del proyecto ha aumentado y el porcentaje de documentación ha disminuido. Esto se ha producido por haber incluido mucho nuevo código pero no aplicando las buenas prácticas de programación en cuanto a la documentación se refiere.

## 2. Plan de mejora

En este nuevo análisis seguimos teniendo una clasificación SQALE global de D, y considero que el proyecto no tiene una mala calidad pero si peor que en análisis.

Seguimos teniendo esta D en confiabilidad y seguridad por un fallo crítico en cada categoría que también aparecían en el anterior análisis y no han sido corregidos, cosa que habría que hacer ya que el tiempo estimado es pequeño y ascenderíamos a una clasificación C.

Además tenemos 22 APIs públicas que no han sido comentadas. Esto debe ser solucionado de manera que tengan una documentación asociada para facilitar el trabajo a futuros implicados en el proyecto.

Analizándolo desde el punto de vista de las funciones, debemos reestructurar la función que posee una complejidad de 30, ya que es un valor muy alto para un único método.

Como prioritario considero que habría que solucionar ese par de errores críticos que nos decrementan la calificación, así como los errores mayores ya que también tienen bastante importancia. Obviamente estaría muy bien poder solucionar los issues menores pero no son tan perjudiciales para el proyecto como los citados anteriormente, por lo que considero que se pueden dejar un poco de lado por el momento.