UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO

FACULTAD DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA, INFORMÁTICA Y MECÁNICA

ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA Y DE SISTEMAS

Informe N°1 - Tercer Parcial:

"Análisis de Code Smells"

DOCENTE: ROXANA LISETTE QUINTANILLA PORTUGAL

ALUMNOS:

- Cordova Castro, Marko Leugim 160890
- Guevara Ferro Cristian Luis 171061
- Luna Ccasani, Charlie Joel 161368
- Mallqui Apaza, Nadiabeth Diana 171063
- Quispe Leon,Widmar Raul 171259
- Rojas Cahuana Etson Ronaldao 124821
- Sullca Peralta, Melanie Indira 171070
- Rodriguez Hancco, Rudy Rodrigo 171068

CUSCO – PERÚ

2021

Análisis de “Code Smells” basado en la lista de Martin Fawler. (Frontend)

Smell	Descripción	Análisis
Código Duplicado	Consiste en páginas iguales o muy similares en diferentes fragmentos de la misma base de código.	Si hay código duplicado.
Método Largo / Función Larga	Método / función muy grande y, por lo tanto, difícil de comprender, ampliar y modificar. Es muy probable que este método tenga demasiadas responsabilidades, dañando uno de los principios de un buen diseño de OO (SRP: Principio de responsabilidad de Simple).	No tenemos funciones largas.
Clase Grande	Clase que tiene muchas responsabilidades y por lo tanto contiene muchas variables y métodos. El mismo SRP también se aplica en este caso.	No tenemos clases grandes, todo se resume.
Lista de Parámetros Larga	Lista extensa de parámetros, lo que dificulta su comprensión y suele ser una indicación de que el método tiene demasiadas responsabilidades. Este smell tiene una fuerte relación con el método largo.	No tenemos listas extensas de parámetros.
Lista Divergente	Una sola clase debe cambiarse repetidamente por diferentes razones. Esta es una clara indicación de que no es lo suficientemente cohesiva y debe ser dividida.	No tenemos ese tipo de métodos.
Cirugía de Escopeta	Cuando ocurre una modificación, se deben cambiar varias clases diferentes.	No tenemos ese tipo de métodos.
Característica de Envidia	Cuando un método está más interesado en miembros de otras clases que en la propia, es una señal clara de que está en la clase incorrecta.	No tenemos ese tipo de métodos.
Agrupaciones de datos	Estructuras de datos que siempre aparecen juntas, y cuando uno de los elementos no está presente, todo el conjunto pierde su significado.	Si, en algunos componentes.
Obsesión primitiva	Representa la situación en la que se utilizan tipos primitivos en lugar de clases ligeras.	No
Estructura Switch/ Estructura Switch usada repetidamente	No son necesariamente smells por definición, pero cuando se utilizan ampliamente, suelen ser una señal de problemas, especialmente cuando se utilizan para identificar el comportamiento de un objeto basándose en su tipo o clase.	Si utilizamos switch pero no para identificar el tipo o la clase de un objeto.
Jerarquías de herencia paralelas	Existencia de dos jerarquías de clases totalmente conectadas, es decir, al añadir una subclase en una de las jerarquías, se requiere que se cree una subclase similar en la otra.	No, no existe jerarquía de herencia paralelas.
Clase perezosa	Existencia de clases que no tienen suficientes responsabilidades y por lo tanto no deberían existir.	No, todas las clases tienen una función por lo tanto no poseemos clases perezosas

Generalidad especulativa	Los fragmentos de código están diseñados para soportar futuros comportamientos de software que aún no son necesarios.	Si, nuestro software puede soportar futuros comportamientos.
Campo temporal	Solo para miembros se usa en situaciones específicas, y que fuera de él no tiene significado.	No, la mayoría de nuestras funcionalidades son diseñadas de manera genérica.
Cadenas de mensajes	Un objeto accede a otro, para luego acceder a otro objeto perteneciente a este segundo, y así sucesivamente, provocando un alto acoplamiento entre clases.	No tenemos acoplamiento de clases.
Hombre en el Medio	Identificó lo mucho que una clase no tiene casi ninguna lógica, ya que delega casi todo a otra clase.	No, no existe hombre en el medio cada clase posee un objetivo y es responsable de su cumplimiento
Intimidad inapropiada	Un caso en el que también se conocen dos clases, lo que caracteriza un alto nivel de acoplamiento.	Si
Clases alternativas con diferentes interfaces	Una clase admite diferentes clases, pero su interfaz es diferente.	No, no tenemos clases alternativas que tengan diferentes interfaces.
Biblioteca de clases incompleta	El software utiliza una biblioteca que no está completa y, por lo tanto, se requieren extensiones a esa biblioteca.	No, ya que la biblioteca utilizada está completa.
Clase de datos	La clase sirve sólo como contenedor de datos, sin ningún comportamiento. Generalmente, otras clases son responsables de manipular sus datos, que es un caso de Feature Envy.	Si
Legado rechazado	Indica que una subclase no usa datos o comportamientos heredados.	No
Comentarios	No puede considerarse un smell por definición, pero debe usarse con cuidado ya que generalmente no son necesarios. Siempre que sea necesario insertar un comentario, vale la pena verificar si el código no puede ser más expresivo.	No realizamos comentarios innecesarios.
Nombre misterioso	Nombres no significativos que no representan los elementos del software.	No tenemos clases con nombres insignificativos.
Datos globales	Se puede modificar desde cualquier lugar de la base del código y no hay ningún mecanismo para descubrir qué fragmento de código lo tocó.	Si , existen maneras de modificar datos globales pero también, tenemos métodos que nos permiten identificar

		qué fragmento lo modifico.
Datos mutables	Los cambios en los datos a menudo pueden tener consecuencias inesperadas y errores engañosos.	No existen datos mutables.
Elemento perezoso	Elementos de software diseñados para crecer, pero que no se ajustan a la evolución del software.	No
Tráfico de información privilegiada	Problemas de acoplamiento causados por datos comerciales entre módulos.	Si

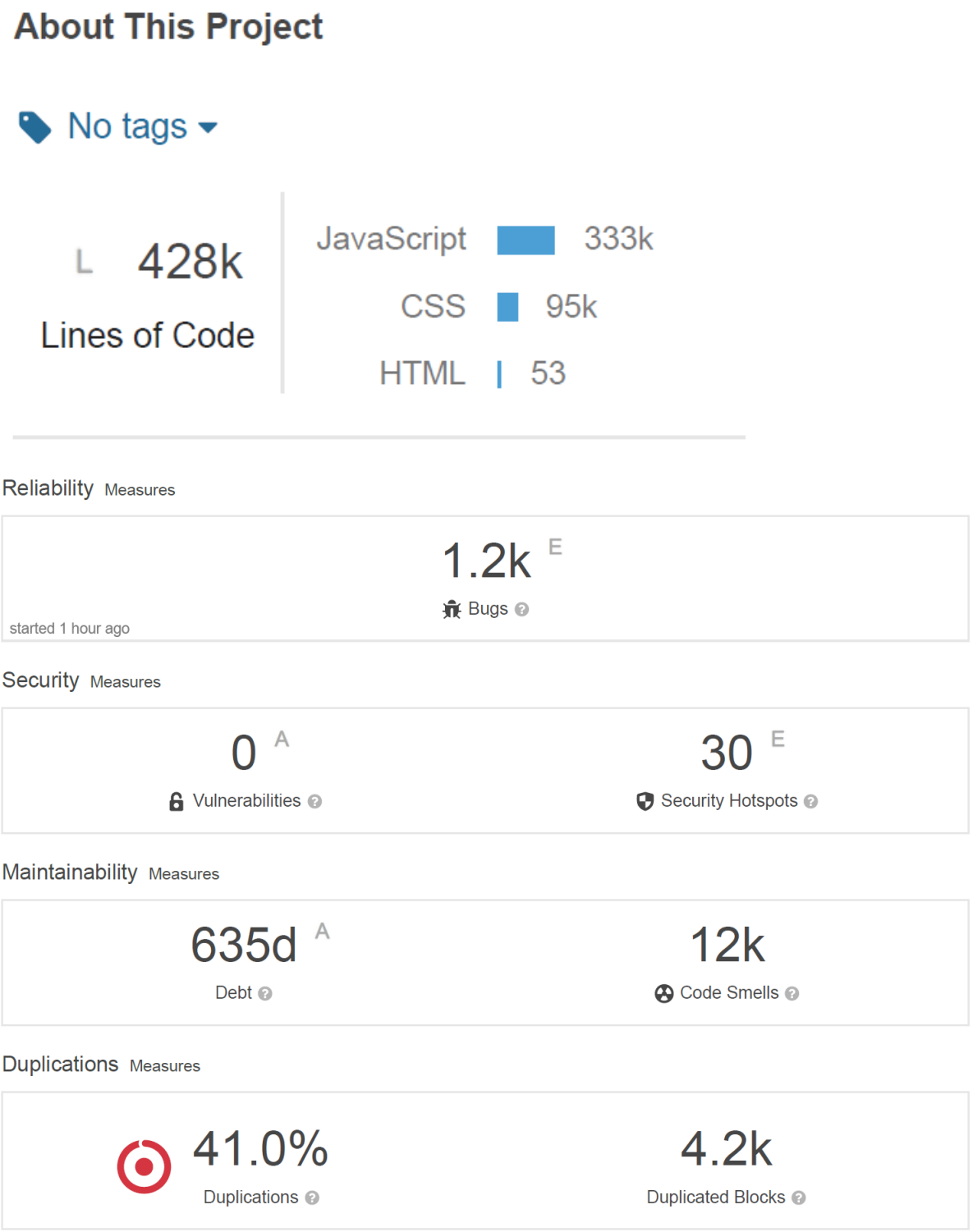
Análisis de "Code Smells" basado en la lista de Martin Fawler. (Backend)

Smell	Descripción	Análisis
Código Duplicado	Consiste en páginas iguales o muy similares en diferentes fragmentos de la misma base de código.	Si hay código duplicado/similar en los models y resources.
Método Largo / Función Larga	Método / función muy grande y, por lo tanto, difícil de comprender, ampliar y modificar. Es muy probable que este método tenga demasiadas responsabilidades, dañando uno de los principios de un buen diseño de OO (SRP: Principio de responsabilidad de Simple).	No existen métodos muy largos.
Clase Grande	Clase que tiene muchas responsabilidades y por lo tanto contiene muchas variables y métodos. El mismo SRP también se aplica en este caso.	Si existe una clase grande (Tutoring Program) requerida en gran mayoría de las otras clases
Lista de Parámetros Larga	Lista extensa de parámetros, lo que dificulta su comprensión y suele ser una indicación de que el método tiene demasiadas responsabilidades. Este olor tiene una fuerte relación con el método largo.	No se tienen métodos que contengan muchos parámetros.
Lista Divergente	Una sola llamada debe cambiarse por muchas razones. Esto es una clara indicación de que no es lo suficientemente cohesivo y debe dividirse.	No porque nuestros models no están estructurados de esa forma.
Cirugía de Escopeta	Cuando ocurre una modificación, se deben cambiar varias clases diferentes.	Si, existen muchas dependencias en los models y resources, si se realiza un cambio en los parámetros, otras clases se ven afectadas.
Característica Envidia	Cuando un método está más interesado en miembros de otras clases que en la propia, es una señal clara de	No hay dependencias fuertes de esa categoría.

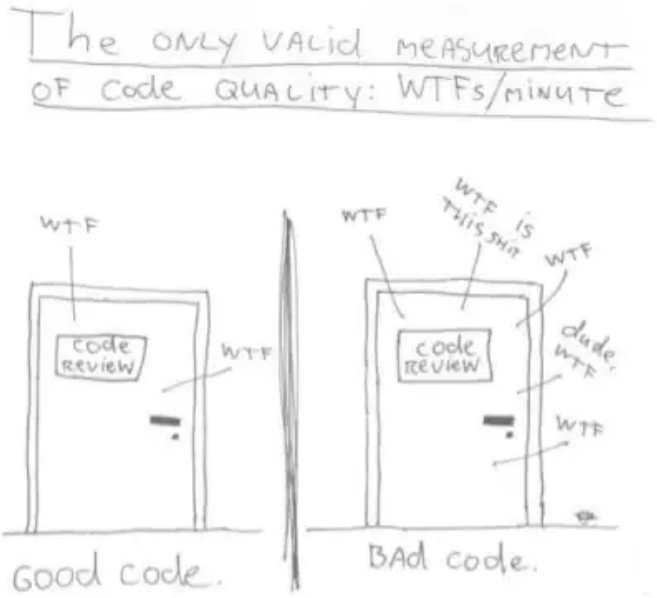
	que está en la clase incorrecta.	
Agrupaciones de datos	Estructuras de datos que siempre aparecen juntas, y cuando uno de los elementos no está presente, todo el conjunto pierde su significado.	Si, porque nuestros datos están relacionados uno con el otro.
Obsesión primitiva	Representa la situación en la que se utilizan tipos primitivos en lugar de clases ligeras.	Se hace uso de clases ligeras y estándar, no de primitivas.
Estructura Switch/ Estructura Switch usada repetidamente	No son necesariamente smells por definición, pero cuando se utilizan ampliamente, suelen ser una señal de problemas, especialmente cuando se utilizan para identificar el comportamiento de un objeto basándose en su tipo o clase.	No, porque cada models y resources están estructurados de forma independiente.
Jerarquías de herencia paralelas	Existencia de dos jerarquías de clases totalmente conectadas, es decir, al añadir una subclase en una de las jerarquías, se requiere que se cree una subclase similar en la otra.	No, no hay subclases, por lo tanto no hay jerarquía.
Clase perezosa	Clases que no tienen suficientes responsabilidades y por lo tanto no deberían existir.	Todas las clases implementadas son importantes para el correcto funcionamiento del sistema.
Generalidad especulativa	Los fragmentos de código están diseñados para soportar futuros comportamientos de software que aún no son necesarios.	Si puede crecer de manera moderada de acuerdo a los métodos ya existentes.
Campo temporal	Solo para miembros se usa en situaciones específicas, y que fuera de él no tiene significado.	No, la mayoría de nuestras funcionalidades son diseñadas de manera genérica.
Cadenas de mensajes	Un objeto accede a otro, para luego acceder a otro objeto perteneciente a este segundo, y así sucesivamente, provocando un alto acoplamiento entre clases.	Si, ya que se necesita un verificación de usuario para acceder a las consultas.
Hombre en el Medio	Identificó lo mucho que una clase no tiene casi ninguna lógica, ya que delega casi todo a otra clase.	Cada clase posee su lógica y no puede delegar a otra.
Intimidad inapropiada	Un caso en el que también se conocen dos clases, lo que caracteriza un alto nivel de acoplamiento.	No, porque cada consulta que se realice necesita de una verificación previa del usuario.
Clases alternativas con diferentes	Una clase admite diferentes clases, pero su interfaz es diferente.	Esto concierne al Front-End

interfaces		
Biblioteca de clases incompleta	El software utiliza una biblioteca que no está completa y, por lo tanto, se requieren extensiones a esa biblioteca.	Flask es un Framework que cuando se instala por primera vez viene con numerosas funcionalidades o las tiene casi todas para hacer una app web completa.
Clase de datos	La clase sirve sólo como contenedor de datos, sin ningún comportamiento. Generalmente, otras clases son responsables de manipular sus datos, que es un caso de Feature Envy.	No, cada clase maneja sus propios métodos para que sean utilizados y/o manipulados
Legado rechazado	Indica que una subclase no usa datos o comportamientos heredados.	Cada clase es independiente de la otra, no existen comportamientos heredados.
Comentarios	No puede considerarse un olor por definición, pero debe usarse con cuidado ya que generalmente no son necesarios. Siempre que sea necesario insertar un comentario, vale la pena verificar si el código no puede ser más expresivo.	Se tienen los comentarios necesarios para describir lo que se hace.
Nombre misterioso	Nombres no significativos que no representan los elementos del software.	No, porque todos nuestros datos son validados con anterioridad.
Datos globales	Se puede modificar desde cualquier lugar de la base del código y no hay ningún mecanismo para descubrir qué fragmento de código lo tocó.	Si, porque existen métodos encargados para eso, ya sea desde una clase primordial o desde sus subclases.
Datos mutables	Los cambios en los datos a menudo pueden tener consecuencias inesperadas y errores engañosos.	No, se restringe el tipo de datos que se leen para evitar errores engañosos.
Elemento perezoso	Elementos de software diseñados para crecer, pero que no se ajustan a la evolución del software.	No, porque todos nuestros models y resources manejan métodos, clases diseñadas y módulos
Tráfico de información privilegiada	Problemas de acoplamiento causados por datos comerciales entre módulos.	Depende de qué tipo de consulta es requerida por el usuario, porque existe una verificación al momento de realizar una consulta.

ANALISIS SONARCLOUD



Why?



BIBLIOGRAFÍA

https://refactoring.com/

https://arxiv.org/pdf/2004.10777.pdf

https://sonarcloud.io/projects