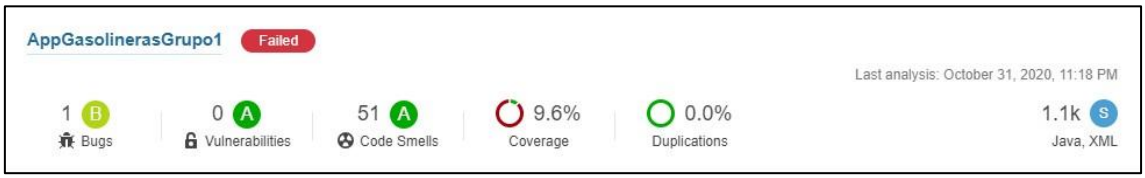
Análisis de Calidad del Producto – Sprint 1

Autores: Miguel Carbayo Fernández y Adrián Celis Fernández

ANÁLISIS 31 OCTUBRE 2020 (Miguel Carbayo y Adrián Celis)

1. Resultados de Sonar



2. Análisis

Como se puede observar el proyecto no llega a los estándares de calidad pedidos debido a que la calificación de *Reliability* (confiabilidad) es de B, cuando tendría que ser A. Esto se produce debido a un *bug* que se encuentra en *MainActivity*.

También nos encontramos con 51 *Code Smells* (problemas de mantenibilidad) con una deuda técnica de 5 horas (Se han añadido 2 horas de deuda técnica en la última integración). Esta deuda técnica se encuentra en: *FiltrosActivity* con 10 *Code Smells* y 50 minutos de deuda, *MainActivity* con 7 *Code Smells* y 35 minutos de deuda, Gasolinera con 2 *Code Smells* y 12 minutos de deuda, *SidebarUITest* con 1 *Code Smells* y 10 minutos de deuda, *TestFiltroTipoGasolinera* con 4 *Code Smells* y 8 minutos de deuda y *PresenterGasolinera* con 2 *Code Smells* y 6 minutos de deuda.

Desde otro punto de vista encontramos 1 fallo bloqueante, 3 críticos, 7 mayores, 21 menores y 19 de info. Como tenemos 5h de deuda técnica necesitamos reducir al menos 1 hora para llegar a tener una buena puntuación.

Cabe destacar que en la clase *SideBarUITest*, el método *sidebarHasAllItems* se encuentra completamente comentado en su interior debido a que el test no pasa con éxito, y no se ha conseguido encontrar solución. Por lo tanto, el *bug smell* bloqueante se ignorará.

3. Plan de acción

1. Lo primero que se tendría que hacer es arreglar el bug que se encuentra en *MainActivity*. Nos restaría 15 minutos de deuda.
2. Arreglar el error crítico de usar *if* en vez de *else if* en la clase *Gasolinera*. (10 minutos)
3. En la clase *MainActivity* se vuelve a crear el *string tipoGasolinera* dentro de un método, cuando ya se ha creado como atributo. (5 minutos)
4. En la clase *TestFiltroTipoGasolina* corregir los 4 *bug smell* intercambiando los argumentos del *assert equals*. (8 minutos)
5. En la clase *MainActiviy* arreglar los errores menores de extraer un trozo de código a un método auxiliar (10 minutos), eliminar un *import* no usado (2 minutos). En la clase *FiltrosActivity* eliminar todos los *imports* no usados (10 minutos).
6. Arreglar todos los fallos menores de la clase *DetailActivity* (41 minutos).
7. En la clase *FiltroActivity* dejar de hacer uso de *getDefaultDisplay()* y *getMetrics()* ya que las dos están *Deprecated* y sustituirlas por *Context#getDisplay()* y *WindowMetrics#getBounds()*

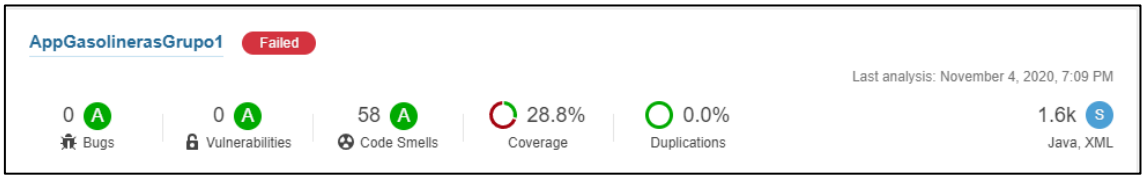
respectivamente como recomiendan las APIS de Android. Arreglar estos dos fallos nos reduciría el tiempo de deuda en 30 minutos.

4. Comentarios

1. Con todas las correcciones descritas obtendríamos una reducción de 3h 26 minutos, y nos quedaríamos con un tiempo de deuda parecido a la original antes de la última integración.
2. Con los pasos 1-6 se puede reducir 1h 41 minutos con los que ya llegaríamos a conseguir reducir la deuda técnica por debajo del máximo de 4 h 10 minutos.

ANÁLISIS 4 NOVIEMBRE 2020 (Adrián Celis)

1. Resultados de Sonar



2. Análisis

Como se puede observar, el proyecto no llega a los estándares de calidad pedidos, ya que nos encontramos con 58 *code smells* (problemas de mantenibilidad) con una deuda técnica de 6 horas y 41 minutos. Esta deuda técnica se encuentra en: *MainActivity* con 26 *code smells* y 3 horas de deuda, *PresenterTarjetaDescuento* con 9 *code smells* y 56 minutos de deuda, *TarjetaDescuento* con 3 *code smells* y 34 minutos de deuda, *BrandExtractorUtil* con 5 *code smells* y 32 minutos de deuda y *PresenterFiltroMarcasTest* con 3 *code smells* y 30 minutos de deuda.

Desde otro punto de vista, encontramos 6 fallos bloqueantes, 7 críticos, 13 mayores, 14 menores y 18 de info. Como tenemos casi 7 horas de deuda técnica, necesitamos reducir al menos 4 o 5 horas para llegar a tener una buena puntuación y permitir un margen de error.

Cabe destacar que en la clase *SideBarUITest*, el método *sidebarHasAllItems* se encuentra completamente comentado en su interior debido a que el test no pasa con éxito, y no se ha conseguido encontrar solución. Por lo tanto, el *bug smell* bloqueante se ignorará.

De la misma manera, sobre la clase *PresenterFiltroMarcasTest* se identifican 3 *code smells* bloqueantes, que se encuentran sin implementar ya que no se ha integrado la rama que contiene esos *tests*, y por lo tanto también se ignorarán estos 3 *code smells.*

3. Plan de acción

1. Lo primero de todo sería conseguir que no tengamos ningún *code smell* de tipo bloqueante por lo que se tendría que modificar la clase *FiltroTipoGasolinaUITest*. En el método *filtroTipoGasolinaTest* se debería añadir algún *Assertion* al caso de prueba. Por otro lado, en la clase *MainActivity,* ya que el método *onOptionsItemSelected* siempre retorna verdadero, no existe ningún caso en el que se devuelva false. Estos cambios reducirían la deuda técnica solo en 14 minutos, pero eliminaríamos los *code smells* bloqueantes que actualmente podemos.

2. Ahora mismo tenemos 7 *code smells* críticos de los cuales 6 pertenecen a la clase *MainActivity*, por lo que lo ideal sería eliminar esos, lo que nos permitiría quitarnos 45 minutos de deuda técnica. Los *code smells* concretos corresponden a:
   - Usar un acceso estático con android.content.*DialogInterface* para BUTTON_POSITIVE ya que los miembros estáticos de una clase base no deberían de ser accedidos haciendo uso de una clase hija. Este problema se repite 3 veces en toda la clase.
   - Los métodos *beforeTextChanged* y *onTextChanged* deberían de llevar un comentario con la explicación de por qué están vacíos, lanzar *UnsupportedOperationException* o contener una implementación. En este caso parece ser un *@Override* para dejar el método vacío, por lo que debería de llevar un comentario de explicación, aunque sea muy simple.
   - Se debería utilizar un acceso estático con *android.content.Context* para el *LAYOUT_INFLATER_SERVICE*, ya que los miembros estáticos de una clase base no deberían de ser accedidos haciendo uso de una clase hija.

   Si se consiguen arreglar estos errores, podríamos obtener una mejora de 30 minutos.

3. Sobre la clase *TarjetaDescuento* tenemos 3 errores mayores que nos ahorrarían 34 minutos. Para corregirlos, el método *newArray* debería devolver un array vacío en vez de retornar nulo. También habría que indicar como *protected* a los constructores de la clase.
4. Ahora actuamos sobre la clase *MainActivity* para intentar reducir al máximo la deuda que genera.
   - Lo primero de todo sería eliminar el código comentado, ya que reduce la legibilidad y el código que no se usa debería ser eliminado.
   - Dejar de usar *setDrawerListener*, en su defecto usar *addDrawerListener*.
   - Cambiar de nombre la variable *test_filtroTipoGasolina* ya que no cumple los convenios de estilo sobre los nombres.
   - Dejar de usar *AsyncTask* ya que esta *Deprecated*. En su defecto se puede usar *Java.Util.Concurrent* y eliminar la importación de la librería de *AsyncTask*.
   - De la misma manera que se deja de usar *AsyncTask*, también se dejan de usar los *.execute* que se usan en *AsyncTask,* se crean *Threads* a los que se les asigna la tarea a realizar y se realiza un *.start* para que ejecuten la tarea.
   - Los métodos *onPreExecute* y *onPostExecute* también se encuentran *deprecated* por ser de una clase (*AsyncTask*) *deprecated*. Lo ideal sería cambiarlos haciendo uso de la concurrencia de Java.

   Después de estos cambios, obtendríamos una rebaja de 1 hora y 52 minutos.

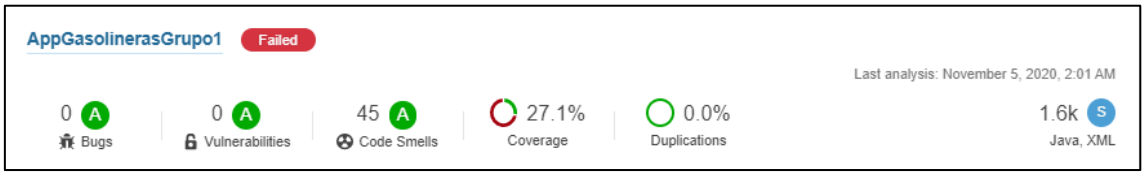
5. Por último, habría que realizar cambios sobre la clase *PresenterTarjetasDescuento*, eliminando primeramente los *imports* que no están siendo utilizados. El método *getListaDeTarjetasDelUsuario* debería de retornar una *List* y no un *ArrayList*. El tipo de *listaDeTarjetasDescuento* debería de ser una *List* en vez de una *ArrayList*. Eliminar *System.out* o *System.err* existentes o sustituirlos por el uso del *logger*. En el método *actualizarListaDePrecios*, la variable g no es usada en ningún momento, con realizar la llamada al método valdría. Estas correcciones nos rebajarían la deuda técnica en 54 minutos.

4. Comentarios

1. Tras realizar todos los cambios, obtendríamos una rebaja de 4 horas y 4 minutos de rebaja permitiéndonos entrar dentro de los límites establecidos de calidad y dejando margen de fallo para próximas implementaciones.
2. Todos los problemas relacionados con *AsyncTask* se podrían solucionar bajando de la API 30 de Android a la API 29 ya que en esa versión no está *Deprecated*.
3. Con realizar los puntos 1, 2, 3 y 5 podríamos obtener el nivel de calidad deseado.

ANÁLISIS 4 NOVIEMBRE 2020 (Adrián Celis)

1. Resultados de Sonar



2. Análisis

Tras estudiar el análisis que realiza *SonarCloud* podemos observar que el código contiene 45 *code smells* agrupados en: 5 bloqueantes, 5 críticos, 7 mayores, 10 menores y 18 de info.

Cabe destacar que en la clase *SideBarUITest*, el método *sidebarHasAllItems* se encuentra completamente comentado en su interior debido a que el test no pasa con éxito, y no se ha conseguido encontrar solución. Por lo tanto, el *bug smell* bloqueante se ignorará.

Esto último ya se comentó en la anterior revisión por lo que sigue siendo necesario para poder analizar el código.

3. Plan de acción

1. Sobre la clase *BrandExtractorUtil* nos encontramos con 5 *code smells* que nos restarían 32 minutos de deuda técnica.

   - El primero consistiría en la eliminación de *imports* que no se utilizan
   - El segundo sería añadirle un constructor privado a la clase ya que las clases de *Utilities* son colecciones de métodos estáticos por lo tanto no están diseñadas para ser inicializadas. Java implementa un constructor implícito a no ser que se le especifique uno, el cual debería de ser privado.
   - El tercero es que el parámetro gasolineras del método *extractBands* debería de realizarse con una implementación de *List* en vez de *ArrayList* ya que a la hora de generar *Utilities* se deben de ayudar a definir una herencia de interfaces de manera que se oculten los detalles de implementación.
   - El cuarto consiste en lo mismo que el tercero, pero en el método *applyFilter*.
   - El quinto indica que el *toLowerCase* y el *equals* se pueden sustituir por el método *equalsIgnoreCase* que realiza la misma funcionalidad.

2. Sobre la clase *MainActivity* se pueden realizar ciertos cambios que consisten en:

- Renombrar la variable *btn_positivo* para que cumpla los estándares de estilo.
- Eliminar código comentado (línea 152).
- Cambiar el método *onOptionsItemSelected* para que no devuelva siempre verdadero.
- Renombrar o reutilizar la variable *adapter* situada en la línea 239 que hace sombra a la misma variable situada en la línea 75.
- Eliminar la especificación del tipo a la hora de crear el nuevo *ArrayList* en la línea 239.
- Exactamente lo mismo que el anterior, pero en la línea 247.
- Lo mismo, pero en la línea 331.
- Eliminar código comentado situado entre las líneas 304 y 307.
- Como ya se pidió en el anterior informe, usar un acceso estático con *android.content.DialogInterface* para *BUTTON_POSITIVE* ya que los miembros estáticos de una clase base no deberían de ser accedidos haciendo uso de una clase hija.
- Como se comentó en el informe previo, realizar comentarios dentro de los métodos *beforeTextChanged* y *onTextChanged* si estos métodos no se van a implementar en ningún momento y se necesitan de esta forma.
- Renombrar o reutilizar la variable *adapter* situada en la línea 423 que hace sombra a la misma variable situada en la línea 75.
- Eliminar la especificación del tipo a la hora de crear el nuevo *ArrayList* en la línea 423.
- En la línea 546 en vez de usar el *if* con la expresión básica de indicar el parámetro res pasado al método, usar *Boolean.TRUE.equals(res).*
- Eliminar casteos innecesarios a *ArrayList* situados en las líneas 548 y 550.
- Con todos estos cambios reduciríamos 1 hora en total de deuda técnica lo que nos permitiría tener un margen de error para posibles implementaciones como ha ocurrido la última vez, ya que la gran mayoría de los errores indicados se corrigieron lo que ha permitido que esta vez el número de fallos no haya añadido una deuda técnica tan alta

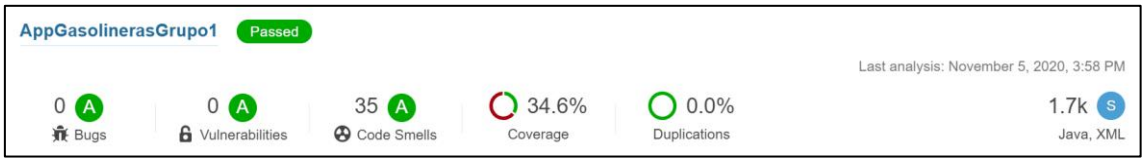
4. Comentarios

1. Con todas las correcciones descritas obtendríamos una reducción de 1h 32 minutos, y nos quedaríamos con un tiempo de deuda por debajo al previo de la implementación previa con los cambios realizados.

2. Fijándonos en los *code smells* bloqueantes nos encontramos con que tenemos una deuda técnica de 30 minutos en la clase *PresenterFiltrosMarcasTest*. Estos *code smells* fueron a ser tratados por el equipo, pero tras el análisis se decidió que la clase iba a ser eliminada o implementada en algún punto posterior, por lo que de momento no se tiene en cuenta sobre el análisis de calidad.

ANÁLISIS 5 NOVIEMBRE 2020 (Miguel Carbayo)

1. Resultados de Sonar



2. Análisis

Como se puede observar el proyecto lega a los estándares de calidad pedidos.

Nos encontramos con 35 *Code Smells* (problemas de mantenibilidad) con una deuda técnica de 3 horas (Se ha añadido 1 hora de deuda técnica en la última integración). Esta deuda técnica se encuentra sobre todo en la clase *MainActivity* (1h 47 minutos).

Desde otro punto de vista encontramos 1 fallo bloqueante, 2 críticos, 15 mayores, 5 menores y 12 de info.

Cabe destacar que en la clase *SideBarUITest*, el método *sidebarHasAllItems* se encuentra completamente comentado en su interior debido a que el test no pasa con éxito, y no se ha conseguido encontrar solución.

3. Plan de acción

1. Para que sonar no salte con un error bloqueante, se añadirá un *assertTrue*(true) para que se solucione el *code smell*. (10 minutos)
2. En la clase *MainActivity* usar un acceso estático para BUTTON_POSITIVE ya que los miembros estáticos de una clase base no deberían ser accedidos por la clase hija. Hay otros casos ya resueltos de este mismo error. (5 minutos)
3. Solucionar los 8 fallos mayores de intercambiar los argumentos en el orden correcto de la clase *PresenterTarjetaDescuentoTest*. (16 minutos)
4. En la clase *MainActivity* cambiar el nombre del *adapter* o dejarlo como atributo. (5 minutos)
5. En la clase Gasolinera quitar el *import* no usado (Log) (2 minutos)
6. En *PresenterFiltroMarcas* y en *PresenterTarjetaDescuento* cambiar el tipo de objeto inicial por *List* en vez de *Arraylist* (solo el tipo de atributo/parámetro, a partir del *new* no cambia) (20 minutos)

4. Comentarios

Tras realizar los cambios obtendríamos una rebaja de 58 minutos, siguiendo por debajo del límite de deuda técnica.