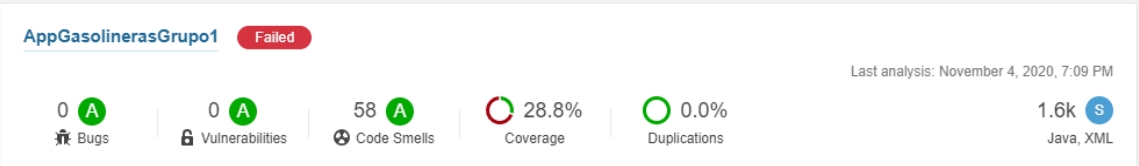
ANÁLISIS 4 NOVIEMBRE 2020:

- Adrián Celis Fernández

CAPTURA:



ANALISIS:

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, randExtractorUtil 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 aproximadamente al menos 4/5 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, los cuales se encuentran sin implementar debido a que no se ha integrado la rama que contiene esos test por lo tanto también se ignorarán estos 3 code smells

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, debido a que en el método filtroTipoGasolinaTest se debería añadir algún Assertion al caso de prueba, y la clase MainActivity debido a que el método onOptionsItemSelected siempre retorna verdadero, no existe ningún caso en el que se devuelva false. Estos cambios reducirían 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 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 porque 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.

-Usar un acceso estático con android.content.Context para 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 ello, el método newArray debería devolver un array vacío en vez de retornar nulo. También habría que indicar como protected a ambos constructores que se encuentran en 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 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 deja de usar los .execute que se usan en AsyncTask y 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 esto 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 se usan. 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.

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.