GIIN 42 CALIDAD DEL SOFTWARE

PRÁCTICA 3 MEJORA DE LA CALIDAD CON SONARQUBE

DANIEL GONZÁLEZ CERVIÑO

ÍNDICE

Enunciado de la actividad

Descripción	
Introducción	Esta sesión de laboratorio está enfocada principalmente en la gestión de calidad del código fuente y de proyectos a través de la herramienta SonarQube (y su versión en la nube, Sonar Cloud). Se aprenderá cómo crear un plan de gestión de calidad en base a Sonarqube llevarlo a cabo.
Objetivo	Aprender la configuración y el manejo de la herramienta SonarQube Ver las posibilidades de Sonarqube para gestionar la calidad del código fuente y los proyectos de desarrollo
Fundamentos	Tema 1, 2, 6 y 7
Entregable	Memoria explicativa (PDF o RTF) de los pasos que se han ido seguido. En la memoria o en la entrega debe constar la dirección del proyecto en GitHub y SonarQube

Introducción

En esta actividad analizamos el proyecto de un compañero desde el punto de vista de la calidad del software.

A continuación proponemos una serie de mejoras que luego implementaremos, para finalmente realizar un pequeño análisis del proyecto antes y después de implementar nuestro plan de calidad.

Datos de acceso

En esta actividad el profesor nos facilita el siguiente proyecto en github https://github.com/Alex-code-01/gestion_agricultura y el siguiente proyecto en sonar cloud https://sonarcloud.io/dashboard?id=Alex-code-01_gestion_agricultura.

Requisito 1

> Analizar los resultados obtenidos.

Proyecto

El proyecto parece ser el resultado de las actividades de la asignatura “Metodología de la programación del alumno”.

El proyecto parece resolver un problema parecido al que yo mismo resolví en este proyecto https://github.com/desarrolla2/viu_08_metodologia_de_programacion.

Sin embargo se detectan algunas carencias.

Documentación

Junto con el proyecto no se ha facilitado ninguna documentación, no hay manual de instalación, no hay tampoco ninguna documentación adicional que ayude a entender el alcance y objetivos del mismo.

Un simple README.md que contenga algunas consideraciones sobre como instalar el proyecto y para qué sirve aportarían una gran ayuda adicional.

Código Limpio

A continuación realizaremos un análisis de la calidad del código desde la perspectiva descrita en *Robert C Martin (2008) Clean Code: A Handbook of Agile Software Craftsmanship* considerado un manual de referencia en esta área.

A continuación destacamos algunos problemas que hemos encontrado y que pueden representar un problema:

Nombres

El autor recomienda evitar el uso de contracciones. En el código se pueden encontrar algunas.

Orientación a objetos

El autor recomienda utilizar no solo la orientación a objetos si no también el cumplimiento de los principios SOLID descritos en un libro anterior el PPP.

El código no utiliza la orientación a objetos y ejecuta una solución procedimental del problema.

Comentarios

El autor recomienda evitar el uso de comentarios, y en el código podemos ver un abuso de los mismos, comentando prácticamente cada línea de código.

Gestión de errores

El autor recomienda separar la lógica del dominio, de la propia gestión de errores, sin embargo el código de este proyecto evita enfrentar la gestión de errores.

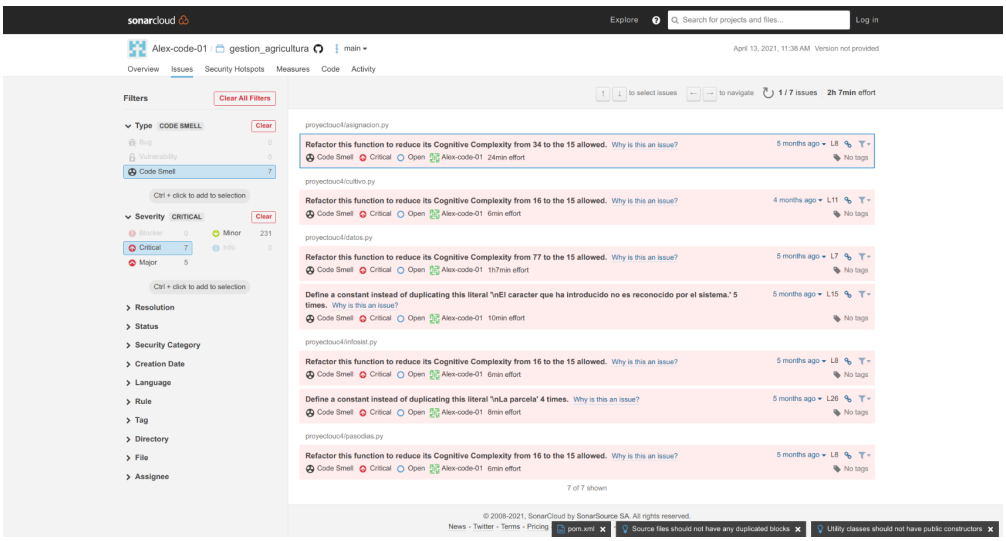
Pruebas automáticas

El autor recomienda utilizar una batería de pruebas, como herramienta de apoyo al desarrollo de software, sin embargo el proyecto no contiene ningún test.

Sonarcloud

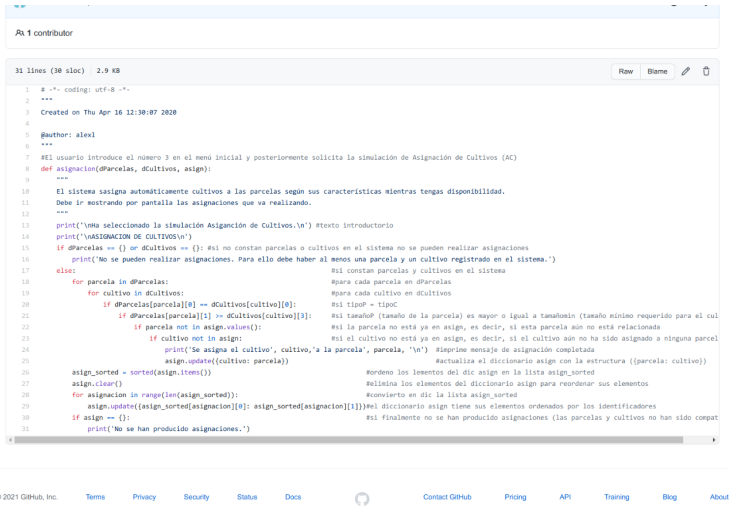
Critical

En sonar cloud el proyecto contiene 7 errores críticos.



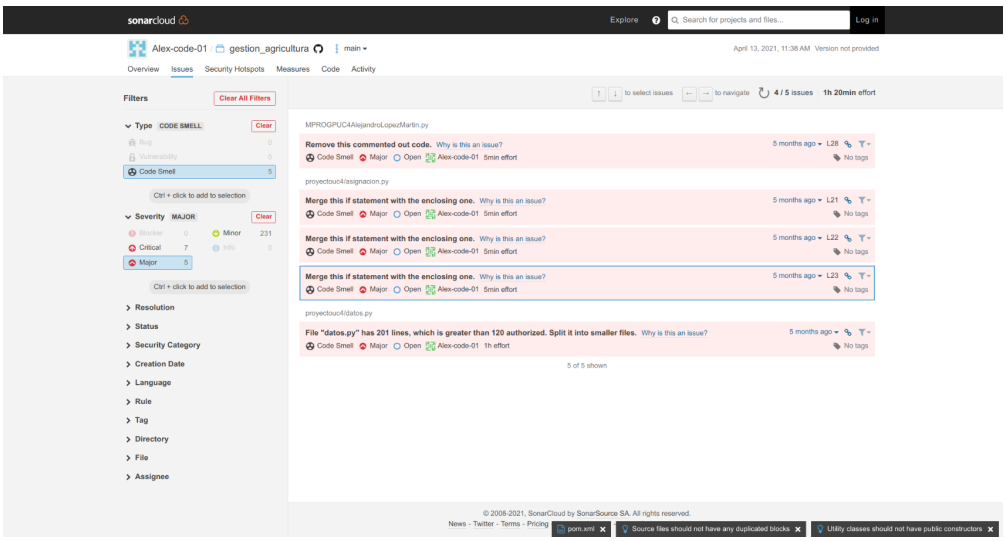
La mayoría de ellos, tiene que ver con la complejidad ciclomática del código y deberían ser resueltos, realizando un mejor diseño del software con funciones más pequeñas y sobre todo con un menor nivel de anidamiento.

Por ejemplo en la siguiente captura podemos ver una función con 6 niveles de anidamiento, cuando el máximo debería ser 2.



Major

El proyecto contiene 5 problemas de tipo mayor.



De nuevo los más destacables están relacionados con el tamaño de las funciones y las clases y con el nivel de anidamiento.

Requisito 2

> Definir un plan de mejora de calidad.

Como hemos dicho anteriormente el proyecto tiene ciertas carencias. Para mejorar la calidad de este proyecto proponemos utilizar la aproximación descrita por *Martin Fowler (1999) Refactoring, improving the design of existing code.*

En este enlace https://docs.google.com/presentation/d/1HLYzPfq5Mz-gT_Llx1ULdO4nJ92zYRerLo74HdU8x_o/edit?usp=sharing puede verse la presentación que preparé para la conferencia "*Refactoring with phpstorm*" en la *Symfony Live 2021*, y que me servirá como referencia para realizar esta actividad.

Escribir una batería de test

Antes de comenzar un proceso de reescritura del código es necesario cubrir el proyecto con una pequeña batería de test que nos permitirá realizar una reescritura del código con seguridad de no romper nada.

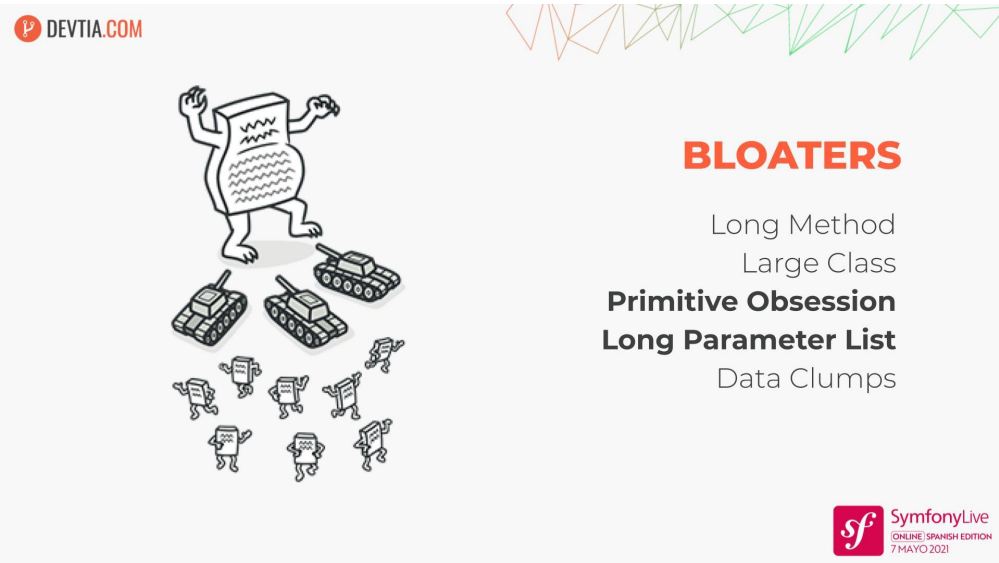
En pequeños pasos

La metodología propuesta por el autor supone enfrentar el problema en pequeños pasos incrementales, dejando siempre el código funcionando tras cada iteración.

Code smells

El siguiente paso es buscar una serie de problemas conocidos y aplicar sobre ellos unas técnicas conocidas como refactors.

En el código del ejemplo probablemente empezaría por aquellos code smells recogidos en el capítulo sobre “gigantismo” y “acoplamiento”.



Requisito 3

> Ejecutar el plan de mejora de calidad.

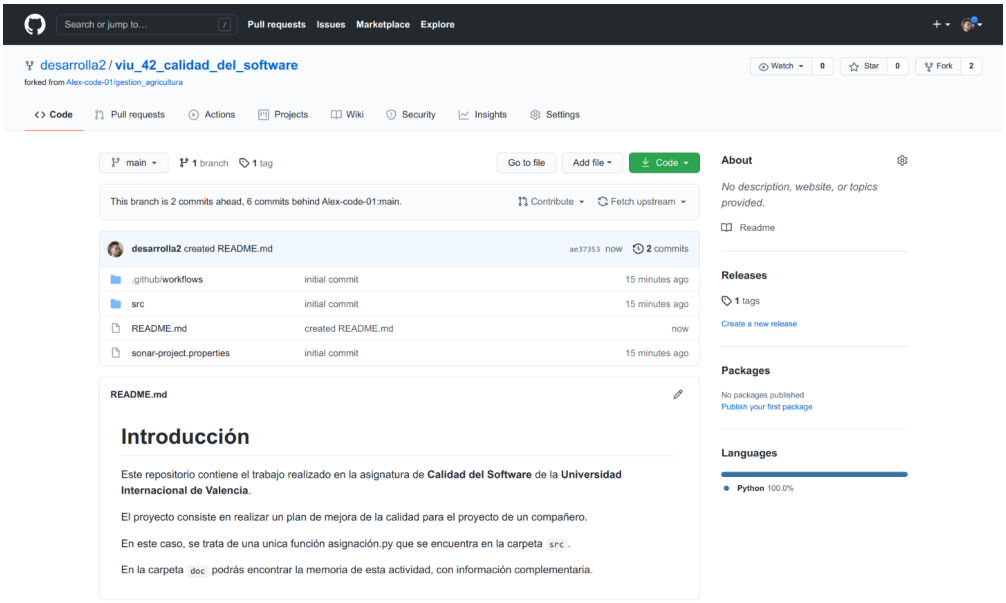
Realmente ejecutar el plan de calidad descrito supone reescribir una parte muy importante del proyecto, lo que probablemente queda fuera del alcance de esta actividad.

Por eso he decidido clonar el proyecto y borrar todo menos una única función.

El proyecto lo he publicado en esta url https://github.com/desarrolla2/viu_42_calidad_del_software y he generado el tag “*initial_commit*” para identificar este estado.

Añadir un fichero README.md

El siguiente paso, como ya digimon es crear el fichero README.md que para indicar la información más relevante del proyecto.

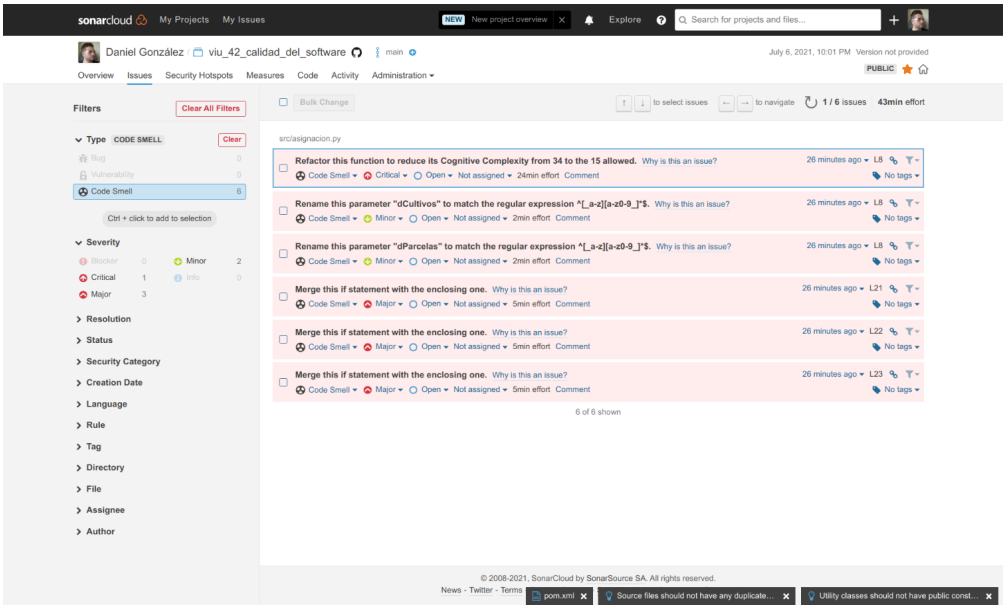


Crear una batería de test

En este punto sería necesario añadir test al proyecto, pero no tengo experiencia con ninguna librería de test en Python por lo que voy a saltar este paso.

Subir el proyecto a Sonar Cloud

Configuramos el proyecto en sonar cloud tal y como lo hicimos en la actividad anterior.

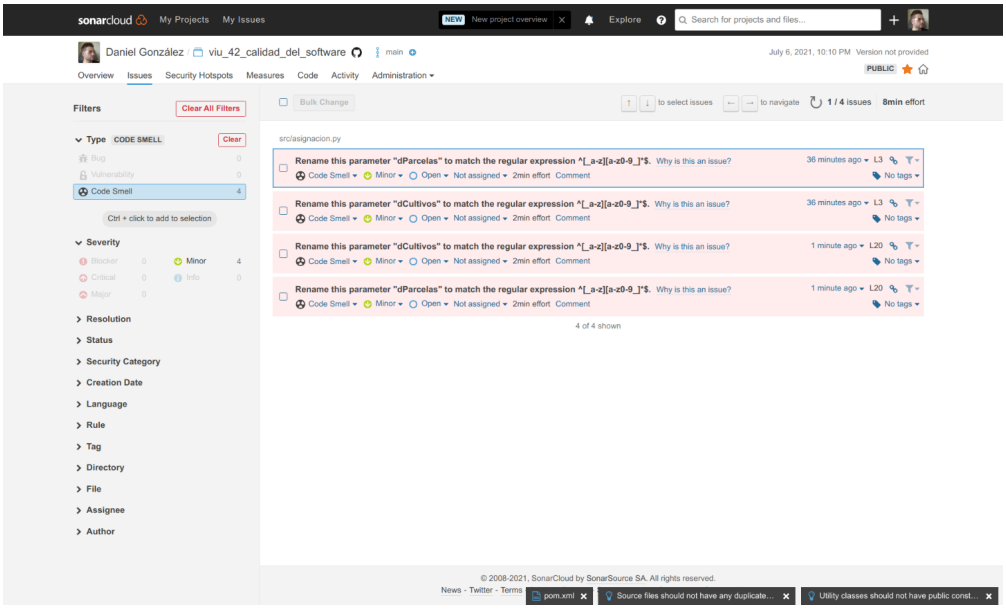


Me reporta 1 Code Smell de tipo *critical* y 3 de tipo *mayor*.

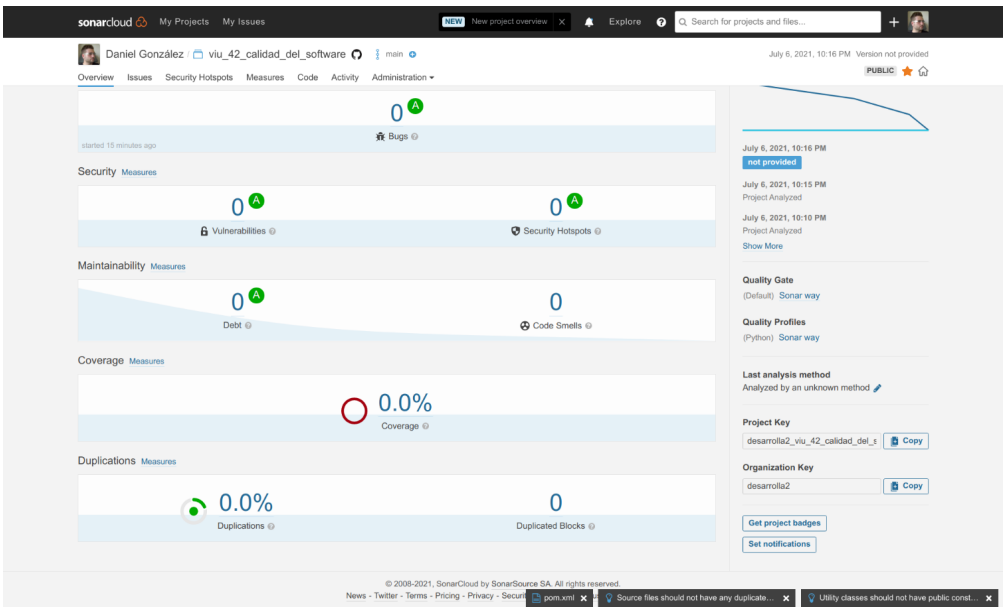
Refactorización del código

Realizamos una reescritura del código extrayendo diferentes métodos, que me permiten ser más explícito en cuanto a la intención del código a la vez que me permiten ocultar los detalles de implementación.

Ejecuto de nuevo sonar cloud, y veo que han desaparecido tanto los *critical* como los *mayor*, pero me sigue indicando que no estoy siguiendo la sintaxis camelcase descrita en la guía de estilo y que las variables deberían seguir una estructura camel case.



Realizó el renombrado de variables y obtengo una puntuación perfecta.



Requisito 4

> Analizar la mejora de calidad entre el ‘antes’ y el ‘después’.

A continuación aparece la versión inicial.

```
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 16 12:30:07 2020

@author: alexl
"""
#El usuario introduce el número 3 en el menú inicial y posteriormente solicita
la simulación de Asignación de Cultivos (AC)
def asignacion(dParcelas, dCultivos, asign):
    """
    El sistema sasigna automáticamente cultivos a las parcelas según sus
características mientras tengas disponibilidad.
    Debe ir mostrando por pantalla las asignaciones que va realizando.
    """
    print('\nHa seleccionado la simulación Asiganción de Cultivos.\n') #texto
introductorio
    print('\nASIGNACION DE CULTIVOS\n')
    if dParcelas == {} or dCultivos == {}: #si no constan parcelas o cultivos
en el sistema no se pueden realizar asignaciones
        print('No se pueden realizar asignaciones. Para ello debe haber al
menos una parcela y un cultivo registrado en el sistema.')
    else:                                                                #si
constan parcelas y cultivos en el sistema
        for parcela in dParcelas:
#para cada parcela en dParcelas
            for cultivo in dCultivos:
```

```
#para cada cultivo en dCultivos
                if dParcelas[parcela][0] == dCultivos[cultivo][0]:          #si
tipoP = tipoC
                    if dParcelas[parcela][1] >= dCultivos[cultivo][3]:      #si
tamañoP (tamaño de la parcela) es mayor o igual a tamañomin (tamaño mínimo
requerido para el cultivo)
                        if parcela not in asign.values():                   #si
la parcela no está ya en asign, es decir, si esta parcela aún no está
relacionada
                            if cultivo not in asign:                        #si
el cultivo no está ya en asign, es decir, si el cultivo aún no ha sido
asignado a ninguna parcela
                                print('Se asigna el cultivo', cultivo,'a la
parcela', parcela, '\n')  #imprime mensaje de asignación completada
                                asign.update({cultivo: parcela})
#actualiza el diccionario asign con la estructura ({parcela: cultivo})
        asign_sorted = sorted(asign.items())
#ordeno los lementos del dic asign en la lista asign_sorted
        asign.clear()
#elimina los elementos del diccionario asign para reordenar sus elementos
        for asignacion in range(len(asign_sorted)):
#convierto en dic la lista asign_sorted
            asign.update({asign_sorted[asignacion][0]:
asign_sorted[asignacion][1]})#el diccionario asign tiene sus elementos
ordenados por los identificadores
        if asign == {}:
#si finalmente no se han producido asignaciones (las parcelas y cultivos no
han sido compatibles)
            print('No se han producido asignaciones.')
```

Y a continuación aparece la versión obtenida después de realizar una reescritura del código corrigiendo los problemas más importantes.

```
# -*- coding: utf-8 -*-

def asignacion(parcelas, cultivos, asign):
    _show_introduccion_message()

    if parcelas == {} or cultivos == {}:
        _show_need_one_pacela_and_one_cultivo_message()
        return

    for parcela in parcelas:
        for cultivo in cultivos:
            _asignate_parcela_to_cultivo(parcelas, cultivos, parcela, cultivo,
asign)

    _sort_and_update(asign)

    if asign == {}:
        _show_no_asignations_message()
```

```
def _asignate_parcela_to_cultivo(parcelas, cultivos, parcela, cultivo, asign):
    if parcelas[parcela][0] != cultivos[cultivo][0]:
        return
    if parcelas[parcela][1] > cultivos[cultivo][3]:
        return
    if parcela in asign.values():
        return
    if cultivo in asign:
        return

    _show_asign_message(parcela, cultivo)
    asign.update({cultivo: parcela})


def _sort_and_update(asign):
    asign_sorted = sorted(asign.items())
    asign.clear()
    for asignacion in range(len(asign_sorted)):
        asign.update({asign_sorted[asignacion][0]:
asign_sorted[asignacion][1]})


def _show_asign_message(parcela, cultivo):
    print('Se asigna el cultivo', cultivo, 'a la parcela', parcela, '\n')


def _show_need_one_pacela_and_one_cultivo_message():
    print('No se pueden realizar asignaciones.')
    print('Para ello debe haber al menos una parcela y un cultivo registrado en
el sistema.')


def _show_introduccion_message():
    print('\nHa seleccionado la simulación Asiganción de Cultivos.\n')
    print('\nASIGNACION DE CULTIVOS\n')


def _show_no_asignations_message():
    print('No se han producido asignaciones.')
```

Un ojo experto encontrará que la implementación después de la reescritura cumple con una serie de principios, de los que ya hemos hablado, que la primera no cumple.

En cuanto a la puntuación en sonar cloud hemos pasado fácilmente de 6 violaciones a 0.

Rúbrica

Calidad de la Memoria. (20%)

> *Redacción impecable, con estructura definida, estilo formal y sin faltas ortográficas. Hay referencias y citas según normativa APA.*

Se han seguido las normas APA

Plan de mejora de la calidad (40%)

> *Se presenta un plan de mejora de calidad adecuado a los problemas detectados, se presenta una selección y priorización de defectos a acometer y justificación de esta.*

Se ha propuesto un plan de mejora de la calidad, siguiendo los principios del libro *Martin Fowler (1999) Refactoring, improving the design of existing code*

Ejecución del plan de mejora de calidad (40%)

> *Se ejecuta de acuerdo con el plan y se reporta una comparativa de estados inicial y final.*

Se ha ejecutado el plan sobre una parte del proyecto original y se han comparado ambos resultados en sonar cloud, obteniendo una puntuación de 0 violaciones en la versión final.