Escuela Técnica Superior de
Ingeniería Informática

Lint Report D04

Group C1.068 | Diseño y Pruebas II | 09/05/2025

Fecha	Versión	Autor
09-05-2025	1.0	Gabriel Vacaro Goytia

Miembros:

- Gabriel Vacaro Goytia (gabvacgoy@alum.us.es)

Repositorio de Github: https://github.com/DP2-C1-068/Acme-ANS-D04

Contenido

Resumen ejecutivo

Este informe presenta un análisis de los *bad smells* detectados por SonarLint en el proyecto Acme-ANS, en concreto, en su versión D04. Se han revisado minuciosamente todas las advertencias generadas, identificando aquellas que pueden considerarse inocuas en el contexto del desarrollo académico. Las incidencias reportadas no afectan a la funcionalidad ni a la mantenibilidad del sistema, y muchas de ellas responden a decisiones de diseño conscientes, acordes con la simplicidad y coherencia del código. El presente documento justifica su permanencia y descarta la necesidad de realizar cambios inmediatos.

Introducción

Este informe presenta el resultado del análisis de código estático realizado mediante la herramienta SonarLint sobre el proyecto Acme-ANS del grupo C1.068. El objetivo principal ha sido identificar los denominados *bad smells*, esto es, patrones o fragmentos de código que podrían considerarse indicativos de malas prácticas o de posibles problemas de calidad, mantenibilidad o legibilidad del sistema.
El informe se ha centrado exclusivamente en los elementos del código propios del proyecto desarrollado, omitiendo cualquier incidencia proveniente del *Acme Framework*, tal como se establece en las directrices docentes. Cada advertencia ha sido revisada de forma individual, valorando su impacto real en el contexto del proyecto, y distinguiendo entre aquellas que deben corregirse y aquellas que pueden considerarse inocuas en el contexto académico actual.
A lo largo del documento se listan todos los *bad smells* considerados inocuos, acompañados de una justificación razonada. Se ha evitado incluir las advertencias que ya han sido corregidas, de forma que el informe refleje únicamente los elementos pendientes que no requieren acción inmediata.
La estructura de este documento es la siguiente: tras la portada y la tabla de revisiones, se incluye el resumen ejecutivo y esta introducción. A continuación, se presentan los *bad smells* con sus respectivas justificaciones. Finalmente, se expone una conclusión con las consideraciones finales y la valoración general sobre el estado del código desde la perspectiva del análisis estático.

Contenido

• **Inyección de campos en lugar de inyección por constructor:** Se utiliza la inyección de campos en todo el proyecto por brevedad y para mantener un patrón consistente. Aunque la inyección por constructor es generalmente más recomendable para facilitar las pruebas y asegurar la inmutabilidad, en el contexto académico y del alcance de este proyecto, la inyección de campos no representa un problema práctico.

• **Literales de texto duplicados (por ejemplo, "status", "aircraft", "maintenanceRecordId"):** Estos literales se repiten de manera controlada en clases específicas. Dado el alcance limitado de cada servicio y la baja probabilidad de que estos valores cambien, la duplicación no afecta negativamente a la mantenibilidad en este contexto. No obstante sería buena idea crear unos *helpers* que ayuden a reducir al máximo el código duplicado.

• **No se ha sobrescrito el método equals:** Entidades como Aircraft, Involves, MaintenanceRecord, Task y Technician extienden a entidades como AbstractEntity o AbstractRole, los cuales ya implementan este método del cual se encarga el framework.

• **Literales booleanos innecesarios:** El uso de literales booleanos señalado no afecta la corrección del código, y su eliminación tendría un efecto meramente estilístico.

• **Sentencias vacías:** Las sentencias vacías se encuentran en métodos que están intencionadamente vacíos ya que no son necesarios para el servicio que lo implementa.

• **Métodos vacíos sin explicación:** Algunos métodos están definidos, pero intencionadamente no implementados, ya que no son necesarios para los casos de uso actuales, podría ser interesante añadir comentarios explicativos.

• **Nombres de paquetes o métodos que no cumplen con el patrón esperado:** En algunos casos los nombres no cumplen estrictamente con la expresión regular recomendada, pero siguen siendo comprensibles, claros y consistentes dentro del proyecto. Cambiar los nombres no aportaría un beneficio significativo.

• **Campos que deberían ser transient o serializables en entidades:** Las advertencias relacionadas con atributos transient o serializables se refieren a datos estadísticos o temporales que no están destinados a persistir en base de datos. Por tanto, se consideran seguros en el contexto actual.

• **Uso de 'assert' en lugar de una comprobación adecuada:** Las aserciones se utilizan como comprobaciones internas durante el desarrollo y no forman parte de la lógica de producción. No afectan al comportamiento en entornos de ejecución reales.

Conclusiones

El análisis realizado con SonarLint ha permitido identificar una serie de *bad smells* en el código del proyecto, los cuales han sido evaluados con criterio técnico y contextual. La mayoría de las advertencias se refieren a prácticas como la inyección de dependencias mediante campos, la repetición controlada de literales, nombres de paquetes o métodos que no siguen convenciones estrictas, o la presencia de métodos vacíos pendientes de implementación. Estas situaciones no afectan negativamente al funcionamiento ni a la estructura del sistema, y en varios casos responden a decisiones conscientes tomadas para simplificar el desarrollo o adaptarse al contexto académico.
No se ha detectado ningún *bad smell* que comprometa la seguridad, la estabilidad o la escalabilidad del sistema. Por tanto, se ha considerado que no es necesario actuar sobre estos elementos en esta fase del desarrollo. No obstante, se recomienda revisar nuevamente estas advertencias en futuras iteraciones, especialmente si el proyecto evoluciona hacia un entorno de producción más exigente.
En definitiva, el informe concluye que el código mantiene un alto nivel de calidad y que las incidencias reportadas no requieren intervención inmediata. Este control contribuye a mantener la trazabilidad y mejora la conciencia sobre la calidad del software durante el desarrollo.

Bibliografía

Web de la universidad de Sevilla - https://ev.us.es