INFORME DE LINT D03

Ignacio Blanquero Blanco

Grupo: C2.026

- Ignacio Blanquero Blanco (ignblabla@alum.us.es)
- Adrián Cabello Martín (adrcabmar@alum.us.es)
- María de la Salud Carrera Talaverón (marcartal1@alum.us.es)
- Joaquín González Ganfornina (joagongan@alum.us.es)
- Natalia Olmo Villegas (natolmvil@alum.us.es)

Repositorio: https://github.com/maryycarrera/Acme-SF-D04

Fecha: 7 de julio de 2024

Tabla de contenido

Resumen ejecutivo 3
Historial de versiones 4
Introducción 5
Informe Lint – Acme-SF-D03 6
Bad Smell 1 6
Bad Smell 2 6
Bad Smell 3 6
Bad Smell 4 6
Bad Smell 5 6
Bad Smell 6 7
Bad Smell 7 7
Bad Smell 8 7
Bad Smell 9 7
Bad Smell 10 7
Bad Smell 11 7
Conclusiones 8
Bibliografía 9

https://github.com/maryycarrera/Acme-SF-D04

Resumen ejecutivo

El propósito de este informe es llevar a cabo un análisis de los “malos olores de código” encontrados en el proyecto Acme-SF-D03 de la asignatura Diseño y Pruebas II. Este proyecto tiene como objetivo mejorar las competencias en el desarrollo web, enfocándose en la implementación de un sistema de información de tamaño moderado. El informe detalla las decisiones adoptadas para la implementación de estos requisitos.

https://github.com/maryycarrera/Acme-SF-D04

Historial de versiones

Versión	Fecha	Descripción
v1.0	06/07/2024	Estructura inicial Informe de Lint D03 y análisis de todos los “bad smells”. Redacción de conclusiones.
v1.1	07/07/2024	Revisión final del documento.

https://github.com/maryycarrera/Acme-SF-D04

Introducción

El presente informe de Lint, titulado "Informe de Lint D03", forma parte de la asignatura Diseño y Pruebas II y está destinado a documentar los “bad smells” encontrados en el código desarrollado. Este proyecto se enfoca en el desarrollo de un sistema de información de tamaño moderado, con el objetivo de mejorar las competencias en el desarrollo web de los integrantes del grupo C2.026.

En el presente informe se van a analizar los “malos olores de código”, que son indicios de posibles problemas en el código que pueden afectar la mantenibilidad y la calidad del proyecto. SonarLint es una herramienta estática de análisis de código que ayuda a identificar estos "bad smells" para mejorar la calidad del software. Este informe documenta los "bad smells" detectados por esta herramienta en el proyecto Acme-SF-D04 y justifica por qué se consideran inocuos.

El contenido del documento se estructura de la siguiente manera: **Resumen ejecutivo**, el cual proporciona una visión general del proyecto y sus objetivos; **Historial de versiones**, que incluye un registro detallado de los cambios y versiones del documento; **Introducción**, que ofrece una descripción general del contenido y la estructura del documento; **Informe de Lint – Acme SF-D03**, que detalla los “bad smells” encontrados tras analizar el código con la herramienta SonarLint; **Conclusiones**, que presenta reflexiones finales sobre el entregable y los aprendizajes obtenidos; y **Bibliografía**, que enumera las referencias utilizadas para el desarrollo del proyecto.

Informe Lint – Acme-SF-D03

Bad Smell 1

Replace this assert with a proper check: assert object != null;

- Justificación: Este assert es un control adicional en tiempo de desarrollo y testing para garantizar que el objeto no sea nulo. En un entorno de producción, las aserciones generalmente están deshabilitadas, por lo que no afectan el rendimiento o comportamiento del código.

Bad Smell 2

Define a constant instead of duplicating this literal "project" 4 times. [+4 locations]: projectId = super.getRequest().getData("project", int.class);

- **Justificación:** La duplicación de literales, aunque no es ideal, no afecta directamente la funcionalidad del código. Definir una constante puede mejorar la mantenibilidad, pero la duplicación en sí misma no causa errores.

Bad Smell 3

DeveloperTrainingModuleCreateService.java: Define a constant instead of duplicating this literal "updateMoment" 4 times. [+4 locations]: super.bind(object, "code", "creationMoment", "details", "difficultyLevel", "updateMoment", "link", "estimatedTotalTime", "project");

- **Justificación:** Similar al caso anterior, la duplicación de literales no compromete el funcionamiento del código. Usar constantes es una buena práctica, pero la duplicación en este contexto es inofensiva.

Bad Smell 4

Use isEmpty() to check whether the collection is empty or not: if (!canBeDeleted && trainingSessions.size() > 0)

- **Justificación:** Aunque `isEmpty()` es más legible y potencialmente más eficiente, el uso de `size() > 0` es una forma válida de verificar si una colección no está vacía. No introduce errores funcionales.

Bad Smell 5

Remove the unnecessary boolean literal: canBeDeleted = trainingSessions.stream().allMatch(trainingSession -> trainingSession.isDraftMode() == true);

- **Justificación:** El boolean literal `== true` es redundante pero no perjudica el funcionamiento del código. Simplificar esta expresión mejoraría la legibilidad, pero su presencia no causa problemas.

Bad Smell 6

Remove this unused "developer" local variable: Developer developer;

- **Justificación:** Una variable no utilizada no afecta el comportamiento del código, aunque puede considerarse código muerto y debe eliminarse para mantener el código limpio.

Bad Smell 7

DeveloperTrainingModuleListMineService.java: Define a constant instead of duplicating this literal "draftMode" 3 times. [+3 locations]: dataset = super.unbind(object, "code", "details", "difficultyLevel", "estimatedTotalTime", "draftMode", "project");

- **Justificación:** La duplicación de literales no afecta la funcionalidad. Definir constantes es una mejora de mantenibilidad, pero la duplicación en sí no introduce errores.

Bad Smell 8

Define a constant instead of duplicating this literal "finishPeriodDate" 4 times. [+4 locations]: super.bind(object, "code", "startPeriodDate", "finishPeriodDate", "location", "instructor", "contactEmail", "link");

- **Justificación:** Similar a los casos anteriores, la duplicación de literales no compromete la funcionalidad del código.

Bad Smell 9

Define a constant instead of duplicating this literal "startPeriodDate" 4 times. [+4 locations]: super.bind(object, "code", "startPeriodDate", "finishPeriodDate", "location", "instructor", "contactEmail", "link");

- **Justificación:** Igual que antes, la duplicación de literales es una cuestión de estilo y mantenibilidad, no de funcionalidad.

Bad Smell 10

DeveloperTrainingSessionCreateService.java: Define a constant instead of duplicating this literal "masterId" 4 times. [+4 locations]: masterId = super.getRequest().getData("masterId", int.class);

- **Justificación:** La duplicación de literales es una mejora de estilo y mantenimiento, pero no afecta el comportamiento del código.

Bad Smell 11

DeveloperTrainingSessionListService.java: Define a constant instead of duplicating this literal "draftMode" 3 times. [+3 locations]: dataset = super.unbind(object, "code", "startPeriodDate", "finishPeriodDate", "location", "instructor", "contactEmail", "link", "draftMode");

- **Justificación:** La duplicación de literales, si bien no es ideal, no afecta el funcionamiento del código y es más una cuestión de estilo.

https://github.com/maryycarrera/Acme-SF-D04

Conclusiones

Los "bad smells" detectados por SonarLint en el proyecto Acme-SF-D04 están principalmente relacionados con la mantenibilidad y la legibilidad del código. Estos "bad smells" incluyen duplicación de literales, uso de aserciones, métodos de comprobación de colecciones, y variables no utilizadas, entre otros. Aunque implementar las sugerencias de SonarLint mejoraría la calidad del código al hacerlo más limpio, mantenible y fácil de entender, la presencia actual de estos "bad smells" no afecta la funcionalidad ni el rendimiento del proyecto, por lo que pueden considerarse inocuos.

https://github.com/maryycarrera/Acme-SF-D04

Bibliografía

- Intencionalmente en blanco.