

Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

Analysis Report



ACME SOFTWARE FACTORY

OUR FIRST PROJECT IN D&T

Grado en Ingeniería Informática – Ingeniería del Software


Diseño y Pruebas 2

Curso 2023 – 2024

Grupo de prácticas: C1-009

Autores por orden alfabético:

Martínez Cano, Juan

	Diseño y Pruebas II Acme-Software-Factory
	Analysis Report

Índice de contenido

1. Resumen ejecutivo	3
2. Tabla de revisiones.....	4
3. Introducción	5
4. Contenido.....	6
5. Conclusiones	9
6. Bibliografía	10

	Diseño y Pruebas II Acme-Software-Factory
	Analysis Report

1. Resumen ejecutivo

En este informe detallaré mi análisis para los requisitos del primer entregable correspondiente al Student 5 que requieran de interpretaciones adicionales.

	Diseño y Pruebas II Acme-Software-Factory
	Analysis Report


2. Tabla de revisiones

Fecha	Versión	Descripción
02/03/2024	1.0	Primera versión del documento.

	Diseño y Pruebas II Acme-Software-Factory
	Analysis Report

3. Introducción

A continuación, detallaré la lista de requisitos abordados junto con las diversas opciones consideradas y la solución seleccionada. También, en ciertos casos, incluiré un enlace al hilo del foro que detalla la decisión adoptada.

	Diseño y Pruebas II Acme-Software-Factory
	Analysis Report

4. Contenido

Para la realización de la tarea:

Task S05-002

Con descripción:

Code audits are essential pieces to ensure the quality of a project. The system must store the following data about them: a code (pattern “[A-Z]{1,3}-[0-9]{3}”, not blank, unique), an execution date (in the past), a type (“Static”, “Dynamic”), a list of proposed corrective actions (not blank, shorter than 101 characters), a mark (computed as the mode of the marks in the corresponding auditing records; ties must be broken arbitrarily if necessary), and an optional link with further information.

Fue necesario realizar un estudio sobre como implementar el atributo “mark” ya que se trataba de un atributo derivado y tenía ciertas dudas respecto al mismo. A continuación dejo constancia de la entrada del foro en la cual me base para tomar la decisión de implementar el atributo “mark” finalmente.

- [Análisis] D02-Student#5 - 002

ADRIANA VENTO CONESA

[Análisis] D02-Student#5 - 002

CONTRAER

Estimado Rafael:

Nos ponemos en contacto con Ud. para solicitarle una aclaración sobre el siguiente requisito de información:

MANDATORY Deliverable D02: data models.

Information requirements.

1. **Code audits** are essential pieces to ensure the quality of a **project**. The system must store the following data about them: a **code** (pattern “[A-Z]{1,3}-[0-9]{3}”, not blank, unique), an **execution** date (in the past), a **type** (“Static”, “Dynamic”), a list of proposed **corrective actions** (not blank, shorter than 101 characters), a **mark** (computed as the mode of the marks in the corresponding auditing records; ties must be broken arbitrarily if necessary), and an **optional link** with further information.

Contexto:

No tengo claro cómo implementar el atributo "mark", ya que parece derivarse de otra entidad de un segundo requisito obligatorio, y necesitaría una validación más compleja, puesto que piden un desempate.

Alternativa 1:


Explicación:

- Computar mark a través de una asociación @OneToMany hacia la entidad de records junto con un método customizado.

Pros:

- Permitiría calcular este atributo.

Contras:

	Diseño y Pruebas II Acme-Software-Factory
	Analysis Report

- La asociación @OneToMany no se considera adecuada según los criterios de la asignatura. Sin embargo, la recomendación (@ManyToOne) no se podría implementar en esta entidad, dado que un "Code Audit" se fundamenta en el análisis de múltiples registros. Además, implicaría una complejidad añadida en la entidad.

Alternativa 2:

Explicación:

- Implementar la lógica para calcular el atributo en la capa de servicio de la aplicación.

Pros:

- Elimina la implementación @OneToMany de la alternativa uno, permitiéndonos calcular el atributo de la misma manera que en la primera alternativa. Además, fomentaría la división de responsabilidades.

Contras:

- Dado que esta casuística aún no ha sido abordada en nuestras clases teóricas, sería necesario posponer la implementación de este requisito.

Solución propuesta:

La segunda alternativa se alinea mejor con las buenas prácticas enseñadas en la teoría de la asignatura. Por lo tanto, si tuviera que elegir, me inclinaría por esa opción.

Saludos, Grupo C1.049.

Respuesta:

Hace 6 días

RAFAEL CORCHUELO GIL PROFESOR ADMINISTRADOR

RE: [Análisis] D02-Student#5 - 002

CONTRAER

Estimada Adriana:


-Gracias por intentar seguir los consejos que les proporcionamos para realizar sus preguntas y realizar un análisis de alternativas.

Recuerde, que no podemos analizar los requisitos de manera incontextual, sino siempre en el contexto del correspondiente documento de elicitación de requisitos y que debemos tener en cuenta todos los requisitos relacionados al menos directamente con el que tratamos. En este caso, todo parece indicar que las auditorías de código ("code audits") son objetos conglomerados y por lo tanto tenemos que tener en cuenta al menos el requisito que describe las partes de que se componen. Se trata del siguiente:

[D02-S05.03] The result of each code audit is based on the analysis of their audit records. The system must store the following data about them: a code (pattern "AU-[0-9]{4}-[0-9]{3}", not blank, unique), the period during which the subject was audited (in the past, at least one hour long), a mark ("A+", "A", "B", "C", "F", or "F-"), and an optional link with further information.

Es decir, nos dicen que una auditoría de código es un conglomerado compuesto por registros de auditoría y que en cada registro se asigna una nota al código; también nos dicen que la nota global de la auditoría de código se calcula como la moda (concepto estadístico que hace referencia al valor más repetido de un conjunto) y que en caso de que haya más de una moda se debe elegir cualquiera al azar. Es decir, una auditoría de código con las notas "A+", "A", "A", "A", "B", "B", "A" tendría claramente una nota global "A", mientras que una con notas "A+", "B", "B", "F", "F" podría tener como nota global tanto "B" como "F"; en el último caso, nos dicen que cualquiera de las dos alternativas da igual, lo que parece un tanto extraño.

Se ha consultado con el cliente y nos aclara que en caso de empate debe elegirse la nota inferior, por lo que el requisito quedaría redactado así:

	Diseño y Pruebas II Acme-Software-Factory
	Analysis Report

[D02-S05-02] Code audits are essential pieces to ensure the quality of a project. The system must store the following data about them: a code (pattern "[A-Z]{1,3}-[0-9]{3}", not blank, unique), an execution date (in the past), a type ("Static", "Dynamic"), a list of proposed corrective actions (not blank, shorter than 101 characters), a mark (computed as the mode of the marks in the corresponding auditing records; ties must be broken *towards the smallest mark*), and an optional link with further information.

En relación con las alternativas que propone Ud.

- A1: no se la puedo recomendar. Aunque técnicamente es posible usar este tipo de asociaciones, en la práctica los problemas van a ser continuos. Es un concepto que no existe ni en Java ni tampoco en las bases de datos relacionales, por lo que el framework las implementa a través del siguiente subterfugio (que les explicaremos en la próxima sesión de teoría dedicada a temas avanzados): sea la asociación 1 A -> * B; dicha asociación se implementa creando una tabla "A", una tabla "B" y una tabla oculta "A_B" con claves ajenas que relacionan pares de objetos en "A" y "B". Fíjese en que seguramente sería mejor introducir una clave clave ajena (foreign key) *oculta* en "B" que apunte hacia la tabla "A", pero el framework utiliza internamente el componente Hibernate para implementar la persistencia y Hibernate lo hace como le he explicado: con una tabla intermedia oculta. La presencia de esa tabla intermedia no supone un gran problema de eficiencia a día de hoy, pero sí conceptual dado que existe y debemos tenerla en cuenta para todo lo que hagamos, pero al ser oculta queda fuera de nuestro control; y es muy fácil meter la pata y acabar borrando datos en la misma por error si no conocemos al 100% los entresijos del API de persistencia a Java. Por este motivo, nosotros no les recomendamos usar nunca este tipo de asociaciones.

- A2: no la entiendo bien. Sea como sea, "CodeAudit::mark" es un atributo derivado que se calcula creando un histograma que asigna a cada nota un contador de apariciones, ordenándolo a la vez por nota y contador y quedándose con la nota más repetida y más pequeña que obtengamos. Eso se tiene que calcular en un servicio. Le adelanto, pero estudiaremos los detalles en la lección L03, que se puede calcular durante el proceso de loading o de unbinding; si no está familiarizada con esta terminología, no se preocupe; forma parte de las explicaciones de la lección L03. Son conceptos sencillos, pero no se pueden implementar en el modelo de datos directamente por limitaciones tecnológicas.

Lo mejor que ahora puede hacer es modelar correctamente el conglomerado "CodeAudit" como algo formado por "AuditRecords" siguiendo nuestras recomendaciones en cuanto al tipo de relación a usar; cómo se calcula su valor es algo que estudiaremos en la lección L03.

Saludos. RC

	Diseño y Pruebas II Acme-Software-Factory
	Analysis Report

5. Conclusiones

La lecutra realizada en el foro ha sido esencial para poder realizar una correcta impementación del atributo “mark”.

	Diseño y Pruebas II Acme-Software-Factory
	Analysis Report

6. Bibliografía

<https://ev.us.es/>