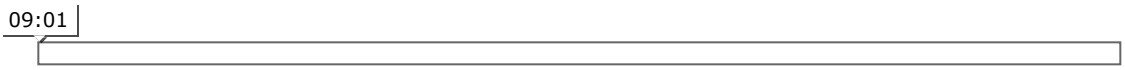
Watch: 🕵️ **Long method: Detección, problemas y estado ideal**



🕵️ Long method: Detección, problemas y estado ideal

El primer code smell que vamos a tratar es precisamente uno de los más habituales y fáciles de identificar: Long method

A la hora de consensuar dentro del equipo de trabajo cuándo podemos señalar que un método es excesivamente largo no debemos olvidar que el propio contexto del proyecto y la tipología de elementos presentes juegan un papel muy importante

Cuando hablamos de distintas tipologías nos estamos refiriendo por ejemplo a la diferencia entre Servicios como este CourseCreator y

Modelos como este Course

- Si os interesa profundizar más sobre podéis consultar aquí los cursos de Arquitectura Hexagonal y Principios SOLID dentro de la plataforma 🙌

Clase `CourseCreator` 👍:

```
// ...
public void create(CourseId id, CourseName name, CourseDuration
duration) {
    Course course = Course.create(id, name, duration);

    repository.save(course);
    eventBus.publish(course.pullDomainEvents());
}
```

En este sentido, entendemos que en términos de **responsabilidad única** este servicio sólo se encargue de hacer una cosa (Huyamos de servicios con sufijos del tipo *service* o *manager* que dan pie a meter de todo en ellos 🏃💨), por lo que debería tener un método lo más conciso posible. Como vemos al entrar en este *create*(), único punto de entrada del servicio, resulta muy sencillo ver a determinado nivel de abstracción qué es lo que sucede dentro sin necesidad de quebrarnos la cabeza

Clase `StudentGradeCalculator` 👎:

```
// ...
public float calculateGrades(final List<Pair<Integer, Float>>
examsGrades, final boolean hasReachedMinimumClasses) {
        if (!examsGrades.isEmpty()) {
            boolean hasToIncreaseOneExtraPoint = false;

            for (Map.Entry<Integer, List<Pair<String, Boolean>>>
yearlyTeachers : allYearsTeachers.entrySet()) {
                if (!(yearToCalculate !=
yearlyTeachers.getKey())) {
                    List<Pair<String, Boolean>> teachers =
yearlyTeachers.getValue();

                    for (Pair<String, Boolean> teacher :
teachers) {
```

```
                        if (teacher.second() != true) {
                            continue;
                        }
                        hasToIncreaseOneExtraPoint = true;
                    }
                } else {
                    continue;
                }
            }
        // more...
    }
```

Por contra, como vemos en este ejemplo, al margen del número de líneas (más de 40 😨) como un factor cuantitativo, destacamos el factor cualitativo, y es que salta a la vista la complejidad existente, que nos obliga a profundizar bastante en el método para entender qué es lo que está haciendo

Y vosotros ¿Tenéis ejemplos de métodos largos? Si os apetece, nos encantaría que los compartierais en un comentario y nos contéis su razón de ser 😬

¿Alguna Duda?

Si tienes cualquier duda sobre el contenido del video o te apetece compartir cualquier sugerencia recuerda que puedes en abrir una nueva discusión justo aquí abajo 👇👇👇

¡Nos vemos en el siguiente video: 🤹 Refactor Extract Method ayudándonos del IDE y problemas relacionados!