INGENIERÍA DEL SOFTWARE

ANÁLISIS DE REQUISITOS 3 – PRUEBAS UNITARIAS

MOHAMED LARGO YAGOUBI
THEO MORENO LOMERO
GERARD NICOLAU FORASTER

2023 – 2024

1. INTRODUCCIÓN

En esta pràctica se nos ha pedido la implementación *Plataforma accesible de Gestión Ciudadana con reconocimiento Biométrico,* en el cual hemos trabajado durante el semestre, hemos desarrollado una versión simplificada del uso *Ejercer derecho al voto.*

Se ha divido en dos partes, la parte más simple del escenerio consistia en la identificación manual del votante, y la otra parte consistia en la identificacion del votante pero utilizando la verificación biometrica.

Durante la implementacíon del projecto se ha utilizado y puesto en práctica metodologías vistas en clase como los principio SOLID, los patrones GRASP y la identificación de code smells.

2. PRINCIPIOS SOLID

Los principios SOLID son 5 fundamentos de diseño orientado a objetos.

Entre ellos, uno que se ha tenido en cuenta es el SRP, el principio de responsabilidad única, que nos indica que una clase debe tener una sola responsabilidad. Donde más se puede observar el empleo de este principio es en la implementación de los doubles. Como se puede observar, se han realizado diferentes clases para cada una de las interfaces, eso, con la finalidad de que cada clase represente cada situación posible al llamar a los métodos de las interfaces. Por ejemplo, para ElectoralOrganism se han dedicado 3 clases; ElectoralOrganismServiceFailedConnection, por si la ejecución falla, ElectoralOrganismServiceIncorrect, por si el NIF pasado es incorrecto, y, ElectoralOrganismServiceOk, por si es correcto. El diseño queda más claro y se ahorran posibles problemas de esta manera.

Hay un principio que ya se cumple con el diseño que se nos pide seguir en la práctica, el cual es el ISP, el principio de segregación de interfaces. La práctica se ha dividido en 2 partes; la primera, que es el caso en que el votante se verifique manualmente, es decir con el pasaporte, y, la segunda, que es cuando se verifica biométricamente. Así pues, este principio nos dice que “Los clientes no deben ser forzados a depender de métodos que no usan” y, para cumplir esto, que segreguemos una interfaz demasiado grande si es necesario. Entonces, esto se ha hecho a partir de las dos formas que hay para verificarse, en vez de hacer una sola interfaz que agrupe todos los métodos dedicados a la verificación, se han implementado dos, una dedicada a la verificación con el pasaporte y la otra a la biométrica.

3. PATRONES GRASP

Los patrones GRASP son 5 principios que describen buenas prácticas de la asignación de responsabilidades a las clases de diseño.

Se ha seguido el patrón “Creador” a la hora de implementar las clases BiometricData y SingleBiometricData, siendo BiometricData la clase creadora. SingleBiometricData representa una clave biométrica y BiometricData agrupa las dos claves biométricas. Entonces, la calse BiometricData, termina teniendo la responsabilidad de crear instancias de SingleBiometricData, ya que guarda las dos claves biométricas con instancias de esta clase.

En este mismo ejemplo también se sigue el patrón de “Bajo acoplamiento”. Un acoplamiento aparece en un diseño entre dos clases o más en el que hay dependencia de una clase a otra, entonces, se dirá que el acoplamiento es bajo si la dependencia entre ellas es mínima, haciendo que los cambios en estas clases tengan un impacto menor. Así pues, entre BiometricData y SingleBiometricData existe acoplamiento, ya que BiometricData depende de la otra clase. Pero, prácticamente el único uso que tiene SingleBiometricData es su constructor, inicializar su atributo keyData, así que en la clase BiometricData se puede trabajar con las instancias de la otra clase y realizar los cambios necesarios sin que tengan un impacto importante, teniendo esto en cuenta, se puede considerar que es un bajo acoplamiento.

4. CODE SMELLS

En el desarrollo de la práctica, no nos hemos encontrado con ningún code smell importante, que haya supuesto un método de refactorización que cambiara gran parte del código. Pero, si que en la implementación de VotingKiosk se han observado imports innecesarios, los cuales eran imports individuales para cada una de las clases de “data” y de las interfaces de “services”, para refactorizarlo, simplemente se han importado con la línia “import data.*”, para importarlos todos a la vez y así poder evitar confusiones.

5. CONCLUSIONES

Ésta ha sido una práctica en la que se ha tenido que diseñar un programa basado en un caso real, el caso de uso de ejercer derecho al voto.

Ha sido diferente programar teniendo en mente un caso realista, se ha tenido que pensar como distribuir los métodos y las clases para cada posible situación que se puede dar en el caso de uso, pero, se ha podido dar un mejor diseño que en anteriores programas a partir del seguimiento de distintos conceptos aprendidos en clase, como algunos de los principios SOLID, los patrones GRASP y la identificación de code smells.

En definitiva, creemos que esta práctica nos contribuirá a realizar diseños mejores y más limpios en futuros programas.