Universidad ORT Uruguay

Facultad de Ingeniería

Bernard Wand Polak

Diseño de Aplicaciones II

Evidencia de la aplicación de TDD y Clean Code

https://github.com/ORT-DA2/164660-185720-234059

Santiago Alvarez – N° Estudiante: 185720

Juan Castro – N° Estudiante: 164660

Marcelo Tilve – N° Estudiante: 234059

Contenido

Uso de TDD y Clean Code

Durante la realización de este proyecto se siguieron las prácticas y consejos del libro Clean Code y la metodología Test Driven Development en los paquetes y clases que así lo ameritaba.

La realización de este proyecto aplicando dichos conceptos llevó a mejorar el diseño, la calidad del código y la facilidad de mantenimiento, ya que ocurre una mejor y mayor separación de responsabilidades, evitando el DRY (Don't Repeat Yourself), por lo que el código se vuelve más legible, más entendible, y por ende, más fácil de cambiar o adaptar a lo que se requiera en un futuro.

Para TDD, se utilizó la estrategia Outside-In, centrándose en verificar que el comportamiento de los objetos es el esperado.

Se aplicaron ciertos conceptos de Clean Code cómo mantener los nombres de clases, métodos, variables claras y mnemotécnicas, de manera de entender qué es lo que se está haciendo en cada punto del código y mantener el mismo legible, al utilizar un buen uso de los nombres para los métodos y funciones.

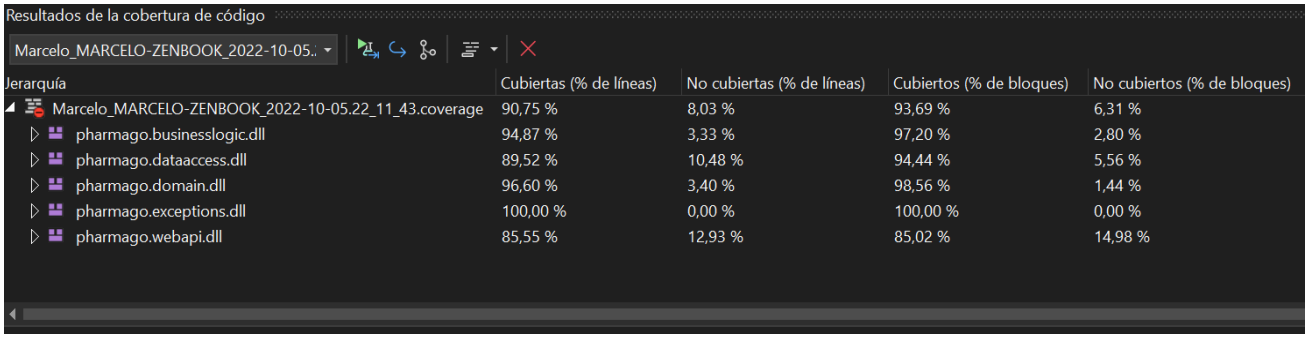
No fue necesario en gran medida realizar comentarios.

Se siguió una secuencia uniforme, haciendo unidades de código pequeñas y con una determinada responsabilidad, evitando comentar en caso de no ser extremamente necesario, siendo consistentes y tratando de evitar la confusión, para que sí en un hipotético caso de que en el futuro un desarrollador nuevo se sumara y debiera mantenerlo, la tarea sea lo menos dificultosa posible.

Resultado de ejecución de las pruebas

Cobertura de las pruebas

Descartando las migraciones que generan un bajo porcentaje en el nivel de cobertura, el ***PharmacyGoDbContext*** y los filtros dentro del proyecto ***WebApi*** se puede asegurar que se cuenta con una elevada cobertura, por arriba del **90%** de las líneas fueron cubiertas de los diferentes proyectos de la solución.



Jerarquía	Cubiertas (% de líneas)	No cubiertas (% de líneas)	Cubiertos (% de bloques)	No cubiertos (% de bloques)
Marcelo_MARCELO-ZENBOOK_2022-10-05.22_11_43.coverage	90,75 %	8,03 %	93,69 %	6,31 %
pharmago.businesslogic.dll	94,87 %	3,33 %	97,20 %	2,80 %
pharmago.dataaccess.dll	89,52 %	10,48 %	94,44 %	5,56 %
pharmago.domain.dll	96,60 %	3,40 %	98,56 %	1,44 %
pharmago.exceptions.dll	100,00 %	0,00 %	100,00 %	0,00 %
pharmago.webapi.dll	85,55 %	12,93 %	85,02 %	14,98 %

Nota: El detalle de los directorios excluidos del code coverage se encuentra en el archivo ***CodeCoverage.runsettings*** dentro del proyecto ***PharmaGo.WebApi.***

Análisis de la cobertura de las pruebas

La cobertura de pruebas unitarias es una métrica porcentual que calcula el grado en que el código fuente de una aplicación fue testeado. Ayuda a determinar la calidad de los tests que se llevan a la práctica y a encontrar partes críticas del código que aún no ha sido probado así como las partes que ya lo fueron.

El análisis del nivel de cobertura fue realizado dentro de ***Visual Studio Enterprise 2022 Versión 17.3.1*** con la herramienta de análisis de cobertura integrada en el mismo.

El nivel de cobertura de pruebas unitarias y de integración del mismo para los paquetes PharmaGo.Exceptions, PharmaGo.Domain, PharmaGo.BusinessLogic, PharmaGo.IBusinessLogic, PharmaGo.DataAccess (sin tener en cuenta archivos de migrations), PharmaGo.IDataAccess, y PharmaGo.WebAPI son elevados y permiten verificar que el código sea correcto.

Esto se debe a la metodología realizada (TDD), a la separación de las clases, a la implementación sencilla de las mismas, y a la calidad y cantidad de las pruebas unitarias, probando cada rama, cada camino del código para corroborar su correcto funcionamiento.

Igualmente, cubrir el 100% de las líneas de código de cada paquete no significa que el mismo no tenga errores o bugs, sino que para esas pruebas realizadas que cubren el 100% de esas líneas de código, las mismas no fallan. Al momento de la entrega, se corrigieron todos los errores o bugs que se hayan conocido previamente.

Debido a la separación de responsabilidades en clases y en esas clases la división de tareas en métodos, los propios métodos y sus respectivas pruebas resultan cortas, legibles, sencillas, de fácil entendimiento y mantenimiento a futuro. De esta manera, se sabe que hace cada método, que prueba cada prueba, con exactitud y sin ambigüedades.