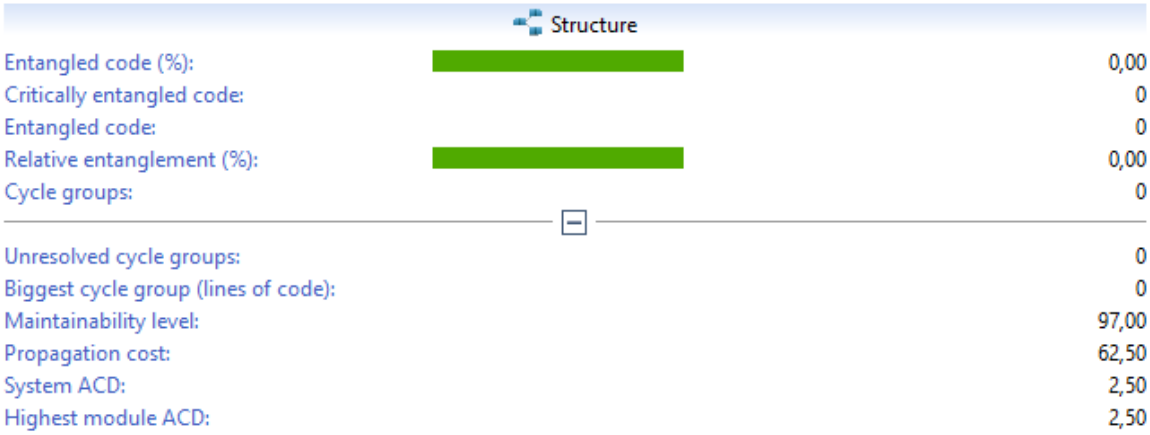
Análisis de acoplamiento – Problema 2

Análisis previo

Al igual que el problema 1, el caso analizado es bastante pequeño (4 clases), por tanto al analizar el panel de resumen de la estructura del proyecto, el ACD, parece alto en proporción; Esto se ve reflejado también en el costo de propagación, que es de más del 60%. En cuanto al nivel de mantenibilidad, es bastante alto, lo cual es bueno.



Al revisar el detalle de cohesión y acoplamiento en el código, se puede ver que todas las clases tienen una cohesión de 3, lo cual es probable que se pueda mejorar mediante la refactorización que se realizará a continuación. En cuanto al acoplamiento, todas las métricas son cero.

El CCD del código base es 10, puntaje que al ser dividido en 4 archivos resulta en un ACD de 2.50.

Solución 1

Modificaciones

Nota: Los cambios explicados a continuación se pueden detallar en este pull request.

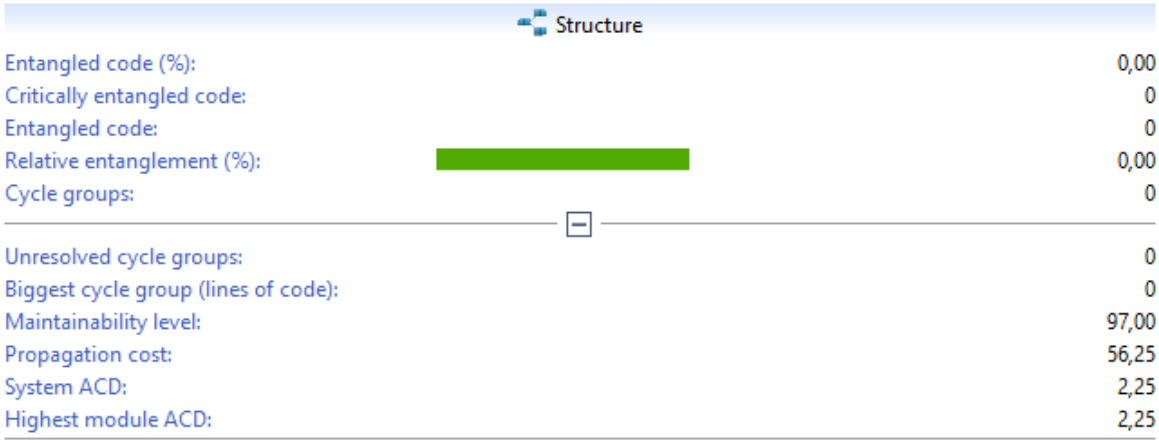
Dado que la clase Inventario estaba funcionando como un singleton pero manejado desde la clase Cajero, la convertí en un Singleton autogestionado. Adicionalmente, modifiqué la firma del método actualizarInventario para recibir como parámetro una lista de productos en vez de un objeto de tipo Carrito, reduciendo el acoplamiento entre estas dos clases y permitiendo al método ser más genérico. Adicionalmente, cambié el nivel de acceso del mapa de productos de público a privado, y creé un nuevo método público para agregar productos a dicho mapa. También removí de esta clase el método darPrecio, el cual fue movido a la clase Carrito explicada a continuación.

En cuanto a la clase Carrito, creé un constructor sin parámetros que inicializa la lista de productos con una nueva lista vacía, para así no tener que inicializar y asignar esta lista en la clase que crea la instancia del Carrito. Al igual que en la clase Inventario, se creó un método público para agregar productos a la lista, y se creó un método para que el carrito sea el encargado de calcular el precio de sus propios productos, ya que este cálculo se encontraba previamente ubicado en la clase inventario, reduciendo de esta manera la cohesión, y no siguiendo el principio de responsabilidad única.

En la clase principal, Cajero, realicé los ajustes necesarios para el correcto funcionamiento tras los cambios descritos anteriormente.

Resultados

Podemos ver que la refactorización tuvo el efecto esperado en el código, ya que el ACD se redujo de 2.5 a 2.25; adicionalmente, el costo de propagación se redujo 6.25 puntos, lo cual indica que se mejoraron los aspectos de alta cohesión y bajo acoplamiento en el código.



Del mismo modo, al ingresar al detalle del acoplamiento por módulos, podemos notar que en las clases Inventario y Carrito se redujo la métrica de cohesión de 3 a 2, resultado de la separación de responsabilidades realizada en la refactorización.

Element [4]	Logical Cohesion (...	Logical Coupling (...
production.Cajero	3	0
production.Producto	3	0
production.Inventario	2	0
production.Carrito	2	0

Finalmente, vemos que el CCD se redujo en una unidad, y al ser constante la cantidad de archivos en comparación al código analizado antes de ser modificado, el ACD también presentó una disminución.

Element [1]	ACD	CCD
Main	2,25	9