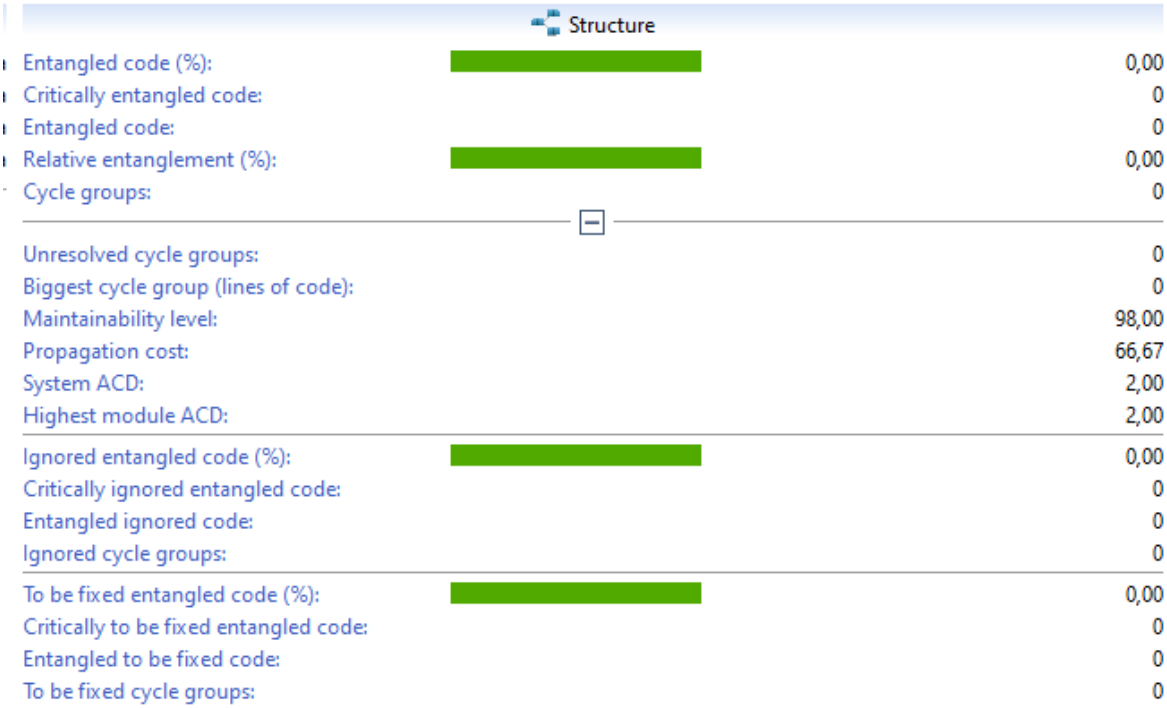
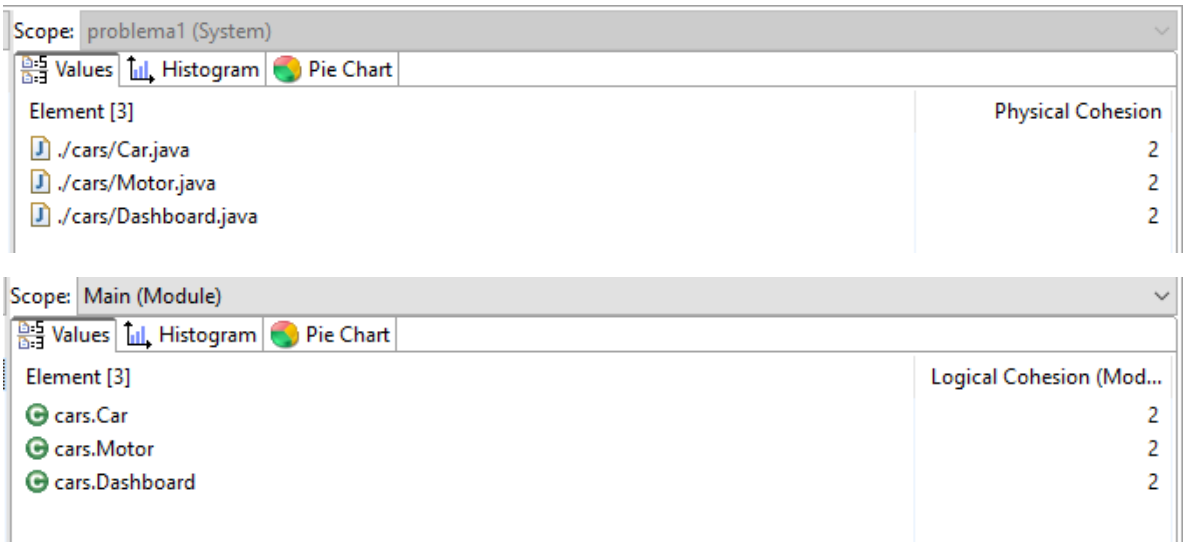
Análisis de acoplamiento – Problema 1

Análisis previo

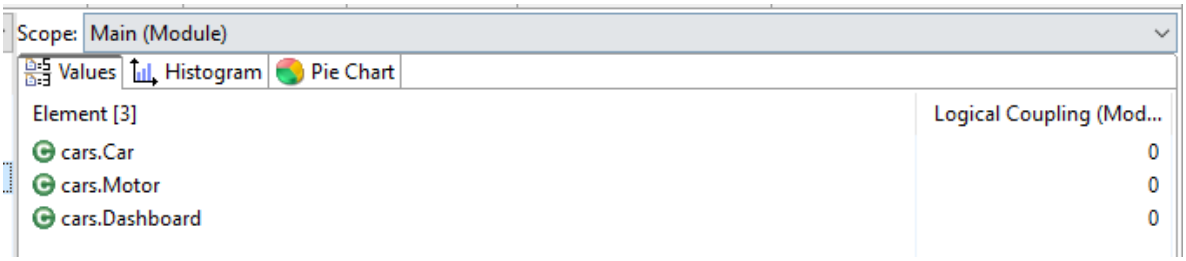
Al realizar el análisis del proyecto en SonarGraph, se encuentra que, al ser un proyecto tan pequeño (solo 3 archivos), el ACD y costo de propagación son bastante altos (2/3). Por otro lado, se encuentra que no hay grupos cíclicos, y el nivel de mantenibilidad es alto (98%).



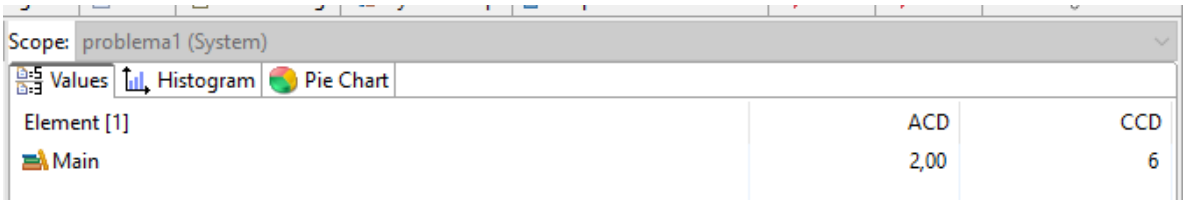
Al revisar en detalle las métricas de cohesión, vemos que, para todos los archivos, la cohesión física es de 2.



Por otro lado, las métricas de acoplamiento muestran cero en todo.



Al revisar el CCD, vemos que el total es 6 (2+2+2), y el ACD es de 2 (6 CCD / 3 archivos).



Solución 1

Modificaciones

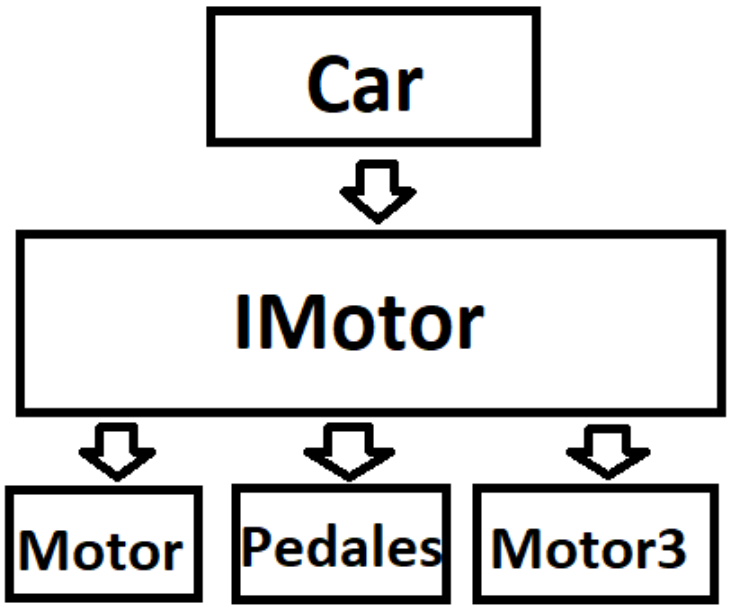
Nota: Los cambios explicados a continuación se pueden detallar en este pull request.

Como un primer acercamiento a la refactorización de este problema, decidí intentar desacoplar módulos mediante el uso de interfaces. En primer lugar, creé una nueva clase Main, para allí alojar la función main, y no dejarla en la misma clase Car (única responsabilidad). A partir de allí, creé una interfaz IMotor para a partir de ésta implementar los motores que pudieran tener los carros a futuro con el posible crecimiento de la aplicación, y una interfaz IPrintable, para ser implementada en los componentes de los cuales se pudiera imprimir información en el Dashboard.

En cuanto al Dashboard, lo desacoplé de la clase Motor, de la cual extraía información de sus campos, lo cual creaba un alto acoplamiento. En lugar de esto, convertí el método printDashboard en estático, y recibe como parámetro la instancia de un objeto que implementa la interfaz IPrintable, de esta manera es mucho más genérico, y con esto, más desacoplado.

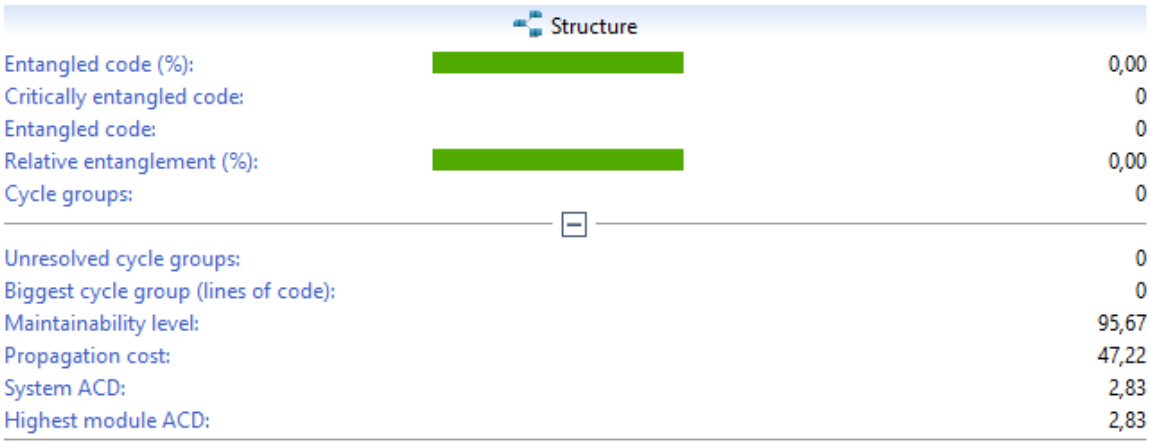
En la solución original, la clase Car era encargada de modificar los campos de la clase motor, rompiendo así el principio de responsabilidad única; así que modifiqué el tipo del campo motor, por la interfaz IMotor, la cual contiene en su firma los métodos Accelerate y Stop, de esta manera es cada clase implementadora de IMotor la encargada de modificar su estado según se llame el método de aceleración o frenado en la clase Car.

En cuanto al constructor de Car, removí el parámetro Dashboard, ya que, tras la refactorización, no es necesario; y modifiqué el parámetro de tipo Motor, por el tipo correspondiente a la interfaz IMotor. Este parámetro decidí conservarlo para que sea posible crear un objeto Car con cualquier objeto que obedezca al contrato de IMotor (como tipo inyección de dependencias), desacoplándolo así de Implementaciones específicas, como se puede ver tal vez con mayor claridad en la siguiente imagen:

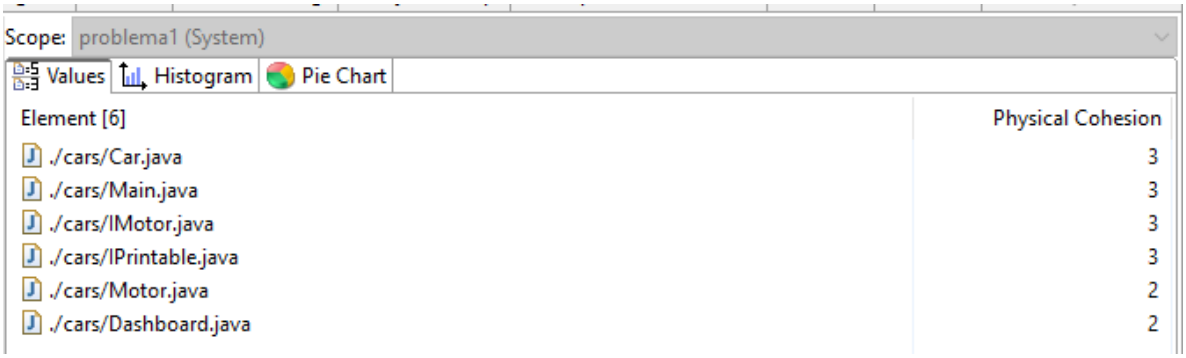


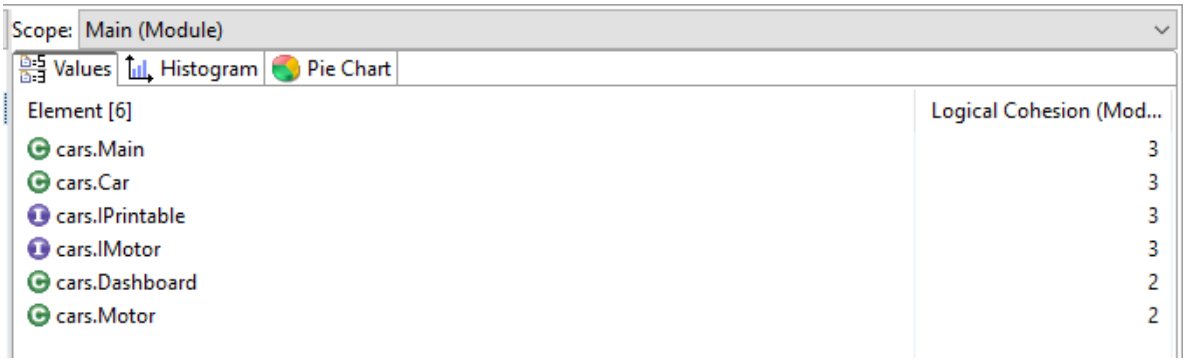
Resultados

Tras realizar estos cambios, vemos que las métricas asociadas a estructura empeoraron levemente, el ACD pasó de 2 a 2.83, y el niel de mantenibilidad bajó poco más de 2%. La única métrica que mejoró fue el costo de propagación, el cual se redujo a 47.22 (previamente era 66.67).

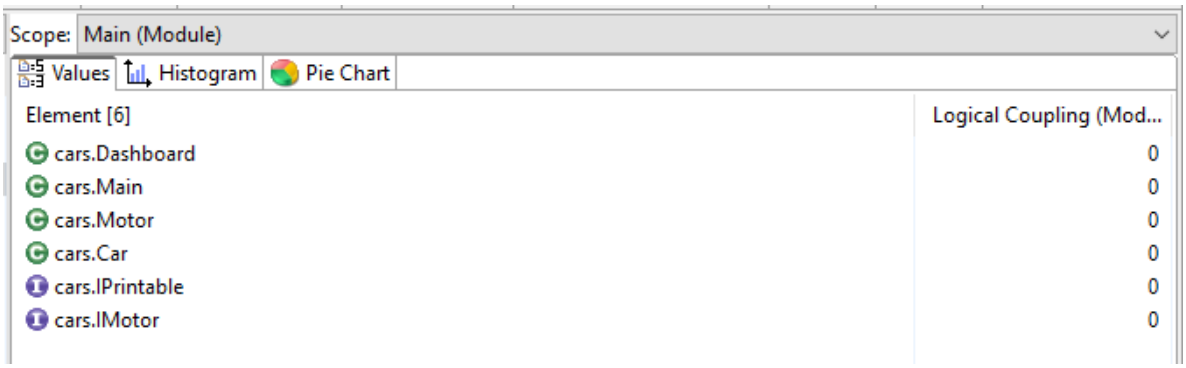


En cuanto a las métricas de cohesión, notamos que Motor y Dashboard se mantuvieron en 2, sin embargo, cabe destacar que debido al uso de las nuevas interfaces, la cohesión de estas clases a futuro se debería ver “amortizada”.

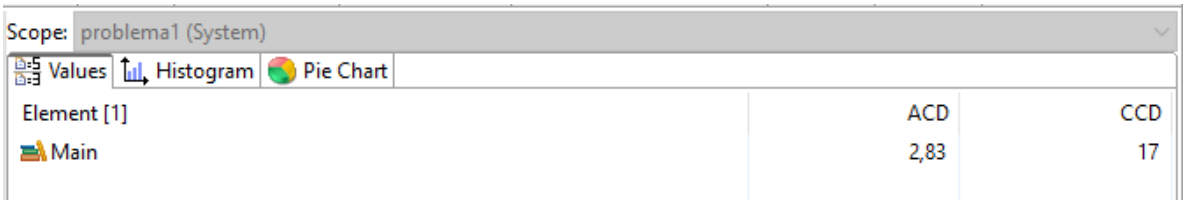




Las métricas relacionadas al acoplamiento se mantuvieron en cero:



Y en cuanto al CCD, se vio aumentado, en parte debido al aumento en la cantidad de archivos, pero al ver el ACD, se denota que el promedio de dependencia también aumentó.



Tras obtener estos resultados inesperados, decidí realizar algunos cambios, reduciendo la cantidad de clases, y centrándome solo en intentar reducir las dependencias entre clases.

Solución 2

Modificaciones

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

Al igual que en la solución 1, moví a motor la lógica relacionada a modificar los campos de motor, para así seguir el principio de responsabilidad única; también, al igual que en la solución 1, creé una clase Main, para desacoplar la función main() de la clase Car.

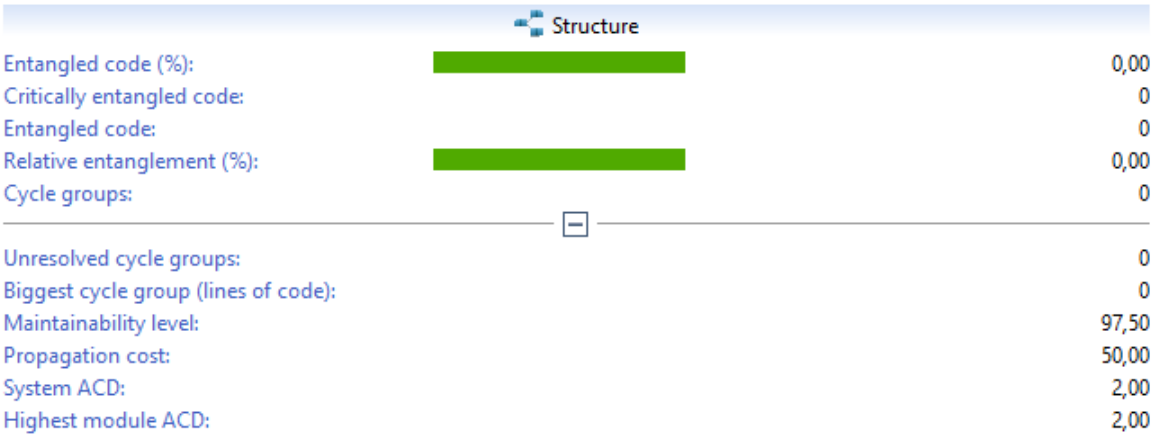
En cuanto a Dashboard, eliminé el campo "motor", y modifiqué el método printDashboard, para que reciba una cadena de texto, y lo muestre en consola, así se retira la dependencia de la clase "Motor".

Por otro lado, en la clase Car, modifiqué el constructor por un constructor sin parámetros, el cual crea allí mismo una instancia de la clase Motor, para así desacoplar esta clase (Motor) de Main.

Tanto en Car como en Motor, se creó el método getStats(), el cual retorna la cadena de texto que será mostrada en pantalla por el Dashboard.

Resultados

Al analizar los resultados obtenidos en la sección resumen de structure, vemos que la única métrica que fue alterada es la de Propagation cost, que se redujo en 16.7%; el resto permanecieron iguales.



En cuanto a las estadísticas de cohesión sí se puede ver una mejora, ya que en las clases Dashboard y Motor se redujo de 2 a 1

Las estadísticas de acoplamiento permanecieron intactas.

El CCD aumentó a 8 debido a la adición de la clase Main, sin embargo, el ACD permaneció en 2.

Reflexión Final

Tras realizar este ejercicio, y a riesgo de estar errado (y basado en que a mi opinión la solución 1 es mejor a la 2, sobre todo pensando en el crecimiento del proyecto), creo que este tipo de métricas no se pueden tomar como verdades absolutas basadas solo en los números, y necesitan algo más de interpretación y experticia, especialmente en proyectos muy pequeños, donde pensar en el bajo acoplamiento “a futuro” puede aparentar que empeora las métricas -contrario a lo que el sentido común indica-.