**SRP: El Principio de Responsabilidad Única**

Nadie más que Buda mismo debe asumir la responsabilidad de revelar secretos ocultos... — E. Cobham Brewer 1810–1897. Diccionario de Frases y Fábulas. 1898.

Este principio fue descrito en el trabajo de Tom DeMarco1 y Meilir Page-Jones2. Lo llamaron cohesión y lo definieron como la relación funcional de los elementos de un módulo. En este capítulo desplazaremos ese significado un poco y relacionaremos la cohesión con las fuerzas que causan que un módulo, o una clase, cambie.

SRP: El Principio de Responsabilidad Única UNA CLASE DEBERÍA TENER SOLO UNA RAZÓN PARA CAMBIAR. Considera el juego de bolos del Capítulo 6. Durante la mayor parte de su desarrollo, la clase Game estaba manejando dos responsabilidades separadas. Estaba llevando un seguimiento del frame actual y estaba calculando la puntuación. Al final, RCM y RSK separaron estas dos responsabilidades en dos clases. La clase Game mantuvo la responsabilidad de llevar el seguimiento de los frames, y la clase Scorer asumió la responsabilidad de calcular la puntuación (ver página 83).

¿Por qué fue importante separar estas dos responsabilidades en clases diferentes? Porque cada responsabilidad es un eje de cambio. Cuando los requisitos cambian, ese cambio se manifestará a través de un cambio en la responsabilidad entre las clases. Si una clase asume más de una responsabilidad, entonces habrá más de una razón para que cambie.

Si una clase tiene más de una responsabilidad, entonces las responsabilidades están acopladas. Los cambios en una responsabilidad pueden afectar o inhibir la capacidad de la clase para cumplir con las demás. Este tipo de acoplamiento conduce a diseños frágiles que se rompen de maneras inesperadas cuando se modifican.

Por ejemplo, considera el diseño en la Figura 8-1. La clase Rectangle tiene dos métodos mostrados. Uno dibuja el rectángulo en la pantalla, y el otro calcula el área del rectángulo.

Dos aplicaciones diferentes utilizan la clase Rectangle. Una aplicación realiza geometría computacional. Utiliza Rectangle para ayudar con las matemáticas de las formas geométricas. Nunca dibuja el rectángulo en la pantalla. La otra aplicación tiene naturaleza gráfica. Puede realizar también geometría computacional, pero definitivamente dibuja el rectángulo en la pantalla.

Este diseño viola el SRP. La clase Rectangle tiene dos responsabilidades. La primera responsabilidad es proporcionar un modelo matemático de la geometría de un rectángulo. La segunda responsabilidad es renderizar el rectángulo en una interfaz gráfica de usuario.

La violación del SRP causa varios problemas desagradables. En primer lugar, debemos incluir la interfaz gráfica de usuario en la aplicación de geometría computacional. En .NET, la interfaz gráfica de usuario tendría que construirse y desplegarse con la aplicación de geometría computacional.

En segundo lugar, si un cambio en la GraphicalApplication hace que el Rectangle cambie por alguna razón, ese cambio puede obligarnos a reconstruir, retestear y redeployar la ComputationalGeometryApplication. Si olvidamos hacer esto, esa aplicación puede romperse de formas impredecibles.

Un diseño mejor es separar las dos responsabilidades en dos clases completamente diferentes, como se muestra en la Figura 8-2. Este diseño traslada las porciones computacionales de Rectangle a la clase GeometricRectangle. Ahora, los cambios realizados en la forma en que se renderizan los rectángulos no pueden afectar a ComputationalGeometryApplication.

¿Qué es una Responsabilidad? En el contexto del Principio de Responsabilidad Única (SRP), definimos una responsabilidad como "una razón para cambiar". Si puedes pensar en más de un motivo para cambiar una clase, entonces esa clase tiene más de una responsabilidad. Esto a veces es difícil de ver. Estamos acostumbrados a pensar en responsabilidades en grupos. Por ejemplo, considera la interfaz Modem en el Listado 8-1. La mayoría de nosotros estará de acuerdo en que esta interfaz parece perfectamente razonable. Las cuatro funciones que declara ciertamente son funciones propias de un módem.

Sin embargo, aquí se están mostrando dos responsabilidades. La primera responsabilidad es la gestión de la conexión. La segunda es la comunicación de datos. Las funciones dial y hangup gestionan la conexión del módem, mientras que las funciones send y recv comunican datos.

¿Deberían separarse estas dos responsabilidades? Eso depende de cómo cambie la aplicación. Si la aplicación cambia de maneras que afectan la firma de las funciones de conexión, entonces el diseño olerá a Rigidez porque las clases que llaman a send y recv deberán recompilarse y desplegarse con más frecuencia de la que nos gustaría. En ese caso, las dos responsabilidades deberían separarse