Consejos para escribir código OO[1]

Estilo de programación

Un buen estilo de programación es una de esas cosas que uno sabe cuando lo ve, pero que es difícil de articular. El canto capitalista –un buen estilo de programación es aquel que produce dinero- es objetivamente medible pero difícil de aplicar día a día.

Hay unas pocas cosas que busco cómo buenos predictores de que un proyecto está en buena forma. Estos elementos son también propiedades que deseo para mi código.

- Una y sólo una vez: Si solamente tuviera un minuto para describir un buen estilo, lo reduciría a una simple regla: un programa escrito con buen estilo todo se dice una vez y sólo una vez. Esto no supone mucha ayuda para escribir buen código, pero es una buena herramienta analítica. Si veo varios métodos con la misma lógica, o varios objetos con los mismos métodos, o varios sistemas con similares objetos, se que esta regla no se cumple. Esto nos lleva a la segunda propiedad:

- Muchas pequeñas piezas: El buen código invariablemente tiene pequeños métodos y pequeños objetos. Sólo mediante la factorización del sistema en muchas pequeñas piezas de estado y función puedes conseguir satisfacer la regla "una y sólo una vez". Observo mucha resistencia a esta idea, especialmente de desarrolladores con experiencia, pero no hay otra cosa que ayude más a los sistemas que dividirlos en muchas piezas. Cuando haces esto, sin embargo, debes asegurarte de que comunicas el cuadro total de forma efectiva. De otro modo, te encontrarás en un plato de "Pasta a la Smalltalk", que es tan desagradable como "Fettucine a la C".

- Reemplazar objetos: El buen estilo lleva a reemplazar objetos fácilmente. En un sistema realmente bueno, cada vez que el usuario dice "quiero hacer que esto sea radicalmente diferente", el desarrollador dice "oh, tendré que hacer un nuevo tipo de X y enchufarlo". Cuando puedas extender un sistema únicamente agregando nuevos objetos sin modificar los existentes, entonces tienes un sistema que es flexible y fácil de mantener. Esto no lo puedes hacer si no tienes muchas pequeñas piezas.

- Mover objetos: Otra propiedad de sistemas con buen estilo es que sus objetos se pueden mover fácilmente a nuevos contextos. Nunca deberías decir "este objeto de este sistema hace lo mismo que aquel objeto de ese otro sistema pero no son exactamente el mismo". Ojo, la primera vez que tratas de mover un objeto descubrirás que lo has hecho mal, que no se generaliza bien. No intentes moverlo demasiado pronto. Lanza un par de sistemas primero. Entonces, si tienes un sistema a base de muchas pequeñas piezas, entonces serás capaz de hacer las modificaciones y generalizaciones necesarios de forma sencilla

- Ratio de cambios: Un criterio simple que utilizo siempre es comprobar los ratios de cambios. Aprendí este criterio de algo que dijo hace tiempo Brad Cox. Desde entonces lo he generalizado: no pongas dos ratios de cambios juntos. No tengas parte de un método que cambia en cada superclase con partes que no cambian. No tengas algunas variables de instancia que cambian cada segundo en el mismo objeto con variables de instancia que

[1] (Smalltalk Best Practice Patterns, Kent Beck)

cambian una vez al mes. No tengas una colección donde los elementos se agregan o eliminan cada segundo con algunos elementos que se agregan o eliminan cada mes. No tengas código en un objeto que cambia por cada pieza de hardware con código que cambia por cada sistema operativo. ¿Cómo evitas este problema? Eso, con muchas pequeñas piezas.

Delegación

Problema: ¿Cómo compartes la implementación sin utilizar la herencia?

Solución: Pasa parte del trabajo del objeto a otro objeto. Otros objetos pueden también delegar a este objeto (o a otra instancia de la misma clase).

Comentario: El principal mecanismo para compartir la implementación es la herencia. Sin embargo a veces necesitamos un objeto que se comporte como otros dos y no tenemos herencia múltiple, en ese caso utilizamos la composición de objetos.

Método Compuesto

Problema: ¿Cómo dividir un programa en métodos?

Solución: Divide tu programa en métodos que realicen una tarea identificable. Mantén todas las operaciones de un método al mismo nivel de abstracción. Esto resultará de forma natural en programas con muchos métodos pequeños, cada uno de unas pocas líneas.

Comentario: Utilizar nombres de mensajes que revelen la intención es un buen motivo para mantener los métodos pequeños. La gente puede leer tus programas más rápidamente y con más precisión si los pueden comprender en detalle, por lo que trocea esos detalles en estructuras de más alto nivel. Dividir un programa en métodos te da una oportunidad de guiar ese troceado. Es un modo de comunicar la estructura de tu sistema.

Los métodos pequeños son más fáciles de mantener. Te permiten aislar asunciones. El código que se ha escrito con métodos pequeños requiere el cambio de unos pocos métodos para corregir o mejorar su operación. Esto es cierto si estás corrigiendo errores, agregando características o ajustando el rendimiento.

Los métodos pequeños también facilitan la herencia. Si decidimos especializar el comportamiento de una clase escrita con métodos largos, a menudo nos encontraremos copiando el método de la superclase y modificando algunas líneas. De ese modo habremos introducido un problema de actualización entre el método de la subclase y el de la superclase. Con métodos pequeños, sobreescribir el comportamiento se convierte en sobreescribir un único método.

Método de Creación Completo

Problema: ¿Cómo representar la creación de instancias?

Solución: Proporciona métodos que creen instancias válidas. Pasa todos los parámetros requeridos a esos métodos.

Comentario: Se trata de evitar que los clientes creen instancias que no estén bien formadas debido a que falta algún parámetro, definir alguna de sus propiades (un punto sin coordenada X o Y, por ejemplo).

Método Consulta

Problema: ¿Cómo escribir un método que comprueba una propiedad de un objeto?

Solución: Devolver un Boolean. Nombrar el método con el prefijo "is" como "isNil"

Comentario: Esta recomendación tiene dos consejos, qué devolver cuando comprobamos una propiedad y cómo nombrar el método. Sobre lo primero, supongamos que tenemos una propiedad "estado", que puede ser "on" u "off". Si queremos comprobar la propiedad y devolvemos la variable estado, el cliente tendrá comprobar si es "on" u "off", de ahí el consejo de devolver un booleano y nos ahorramos esa comparación. El método lo podemos nombrar "isOn" y sirve para comunicar mejor que si se llamara "on" (no se sabe qué hace, si consulta o cambia el valor).

Selector que revela la intención

Problema: ¿Cómo nombramos un método?

Solución: Nombrar los métodos según lo que realizan en vez de nombrarlos según cómo lo realizan.

Usar Mensajes

Problema: ¿Cómo invocar la computación?

Solución: En vez de invocar directamente una rutina que realice una tarea, invoca un nombre y deja que el objeto receptor decida qué hacer con él.

Comentario: Los mensajes son la única estructura de control del sistema. Usamos delegación para que otro objeto haga el trabajo por ti.

Mensaje para alternativas

Problema: ¿Cómo ejecutas una entre varias alternativas?

Solución: Envía un mensaje a uno de entre varios tipos de objetos, cada uno de los cuales ejecuta una alternativa.

Comentario: En vez de preguntar por el tipo de objetos y utilizar sentencias condicionales para resolverlo, se puede utilizar el mismo mensaje que se envía a las diferentes alternativas, cada una de ellas sabe lo que tiene que hacer.