Refactorizar. Proceso de mejora de la estructura interna de un sistema software de forma que su comportamiento externo no varía.

Test o pruebas unitarios. Forma de probar el correcto funcionamiento de un módulo de código. Pruebas a nivel de modulo.

Pruebas funcionales. Tratan a un componente software como una caja negra.

Técnicas de refactorización.

Refactorizaciones simples.

Añadir un parámetro

Motivo: Un método necesita más información al ser invocado
Solución: Añadir como parámetro un objeto que proporcione dicha información
Ejemplo: Cliente.getContacto() →Cliente.getContacto(Fecha f)
Observaciones: Evitar listas de argumentos demasiado largas.

Quitar un parámetro

Motivo: Un parámetro ya no es usado en el cuerpo de un método
Solución: Eliminarlo
Ejemplo: Cliente.getContacto(Fecha f) →Cliente.getContacto()
Observaciones: Si el método está sobrescrito, puede que otras implementaciones del método sí lo usen. En ese caso no quitarlo. Considerar sobrecargar el método sin ese parámetro.

Cambiar el nombre de un método

Motivo: El nombre de un método no indica su propósito
Solución: Cambiarlo
Ejemplo: Cliente.getLimCrdFact() → Cliente.getLimiteCreditoFactura()
Observaciones: Comprobar si el método está implementado en clases base o derivadas.
Si es parte de una interface publicada, crear un nuevo método con el nuevo nombre y el cuerpo del método. Anotar la versión anterior como **@Deprecated** y hacer que invoque a la nueva.
Modificar todas las llamadas a este método con el nuevo nombre.

Refactorizaciones comunes

Mover un atributo

Motivo: Un atributo es (o debe ser) usado por otra clase más que en la clase donde se defi ne.
Solución: Crear el atributo en la clase destino
Observaciones:
Si el atributo es público, encapsularlo primero.
Reemplazar las referencias directas al atributo por llamadas al getter/setter correspondiente.

Mover un método

Motivo: Un método es usado más en otro lugar que en la clase actual
Solución: Crear un nuevo método allí donde más se use.
Ejemplo: Cliente.getLimiteCreditoFactura() →Cuenta.getLimiteCreditoFactura(Cliente c)
Argumento opcional, sólo en caso de necesitar información del cliente
Observaciones:
Es una de las refactorizaciones más comunes. Hacer que el antiguo método invoque al nuevo o bien eliminarlo.
Comprobar si el método está definido en la jerarquía de clases actual. En ese caso puede no ser posible moverlo.

Extraer una clase

Motivo: Una clase hace el trabajo que en realidad deberían hacer entre dos.
Solución: Crear una nueva clase y mover los métodos y atributos relevantes de la clase original a la nueva.
Ejemplo: Sacar datosFactura de cliente
Observaciones:
La nueva clase debe asumir un conjunto de responsabilidades bien definido.
Implica crear una relación entre la nueva clase y la antigua.
Implica 'Mover atributo' y 'Mover método'
Se debe considerar si la nueva clase ha de ser pública o no.

Extraer un método

Motivo: Existe un fragmento de código (quizás duplicado) que se puede agrupar en una unidad lógica.
Solución: Convertir el fragmento en un método cuyo nombre indique su propósito e invocarlo desde donde estaba el fragmento.
Ejemplo: Calcular el salario en función de las horas trabajadas.
Observaciones
Se realiza a menudo cuando existen métodos muy largos.
Las variables locales que sólo se leen en el fragmento se convertirán en argumentos/var. locales del nuevo método
Si algunas variables locales son modificadas en el fragmento, son candidatas a ser valor de retorno del método
Considerar si se debe publicar el nuevo método.

Cambiar condicionales por polimorfismo

Motivo: Una estructura condicional elige entre diferentes comportamientos en función del tipo de un objeto.
Solución: Convertir la estructura condicional en un método ('Extraer método'). Convertir cada opción condicional en una versión del método sobrescrito en la clase derivada correspondiente. Hacer el método en la clase base abstracto.
Observaciones
Con condicionales el código cliente necesita conocer las clases derivadas
/Con polimorfismo el código cliente sólo necesita conocer a la clase base
Esto permite añadir nuevos tipos sin modificar el código cliente
Ejemplo: Calcular el salario en función del tipo de objeto, extraer a un método.

Cambiar código de error por excepciones

Motivo: Un método devuelve un valor especial para indicar un error
Solución: Lanzar una excepción en lugar de devolver ese valor.
Observaciones Decidir si se debe lanzar una excepción verificada o no verificada.
El código cliente debe manejar la excepción.
¡Ojo! cambiar la claúsula 'throws' de un método público (excepciones verificadas) en Java equivale a cambiar la interfaz de la clase donde se define

Refactorización y herencia

Generalizar un método

Motivo: Existen dos o más métodos con idéntico comportamiento (código duplicado) en las clases derivadas.
Solución: Unificarlos en un único método en la clase base
Observaciones
Prestar atención a diferencias en el comportamiento (cuerpo de los métodos)
El código cliente debe manejar la excepción.
Caso particular: un método en clase derivada sobrescribe uno de la clase base pero hace esencialmente lo mismo.
Si el método invoca a métodos declarados en las clases derivadas:
Primero generalizar los métodos invocados si es posible, o declarar dichos métodos como abstractos en la clase base

Especializar un método

Motivo: Un comportamiento de la clase base sólo es relevante para algunas clases derivadas.
Solución: Mover el método que lo implementa a las clases derivadas.
Ejemplo: Empleado.getCuotaVentas() → Vendedor.getCuotaVentas()
Observaciones
Usado en 'Extraer clase derivada'
Puede implicar cambiar la visibilidad de ciertos atributos a protegida (o declarar getter/setter públicos(protegidos) en la base.
Si supone un cambio de interfaz, revisar elcódigo cliente. Otra posibilidad es dejar el método como abstracto en la base

Colapsar una jerarquía

Motivo: Apenas hay diferencias entre una clase base y una de sus derivadas.
Solución: Juntarlas en una sola clase.
Observaciones
Decidir qué clase eliminar (base o derivada)
El código cliente de la clase eliminada (si es pública) deberá ser modificado.

Extraer una subclase

Motivo: Algunas características de una clase son usadas sólo por un grupo de instancias.
Solución: Crear una clase derivada para ese conjunto de características.
Observaciones
Es a menudo consecuencia de usar un diseño top-down
La presencia de atributos como 'tipoCliente' señala la necesidad de este tipo de refactorización.
El código cliente debe ser revisado, en particular, la construcción de objetos de clase base.
Relacionado con 'especializar método', 'reemplazar condicional con polimorfismo'
La política del hospital dice que sólo los médicos titulares pueden tener una carga máxima de pacientes. Los interinos pueden atender un número indefinido de pacientes. Separar en dos clases una que tenga el máximo de pacientes que puede atender.

Extraer una superclase

Motivo: Hay dos (o más) clases con características similares.
Solución: Crear una clase base para ellas y mover el comportamiento común a la base.
Observaciones
A menudo consecuencia de usar un diseño bottom-up.
Casi siempre implica eliminar código duplicado.
La nueva clase base suele ser abstracta o incluso un interfaz.
Hay que prestar especial atención a qué métodos deben subir a la base('generalizar método')
Las referencias en el código cliente pueden tener que ser actualizadas de referencia a derivada a referencia a base

Convertir herencia en composición

Motivo: El principio de sustitución no se cumple: una clase derivada usa sólo parte de la interfaz de la base o no desea heredar también los atributos.
Solución: Crear un campo para la clase base en la derivada, ajustar los métodos involucrados para delegar en la clase base y eliminar la herencia.

Observaciones

Los métodos de la clase base usados en la derivada se implementan en la derivada y se convierten en una invocación al método base (delegación)