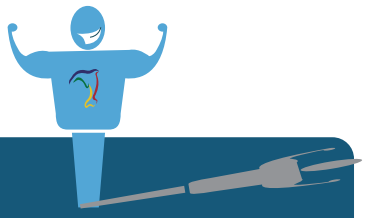
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

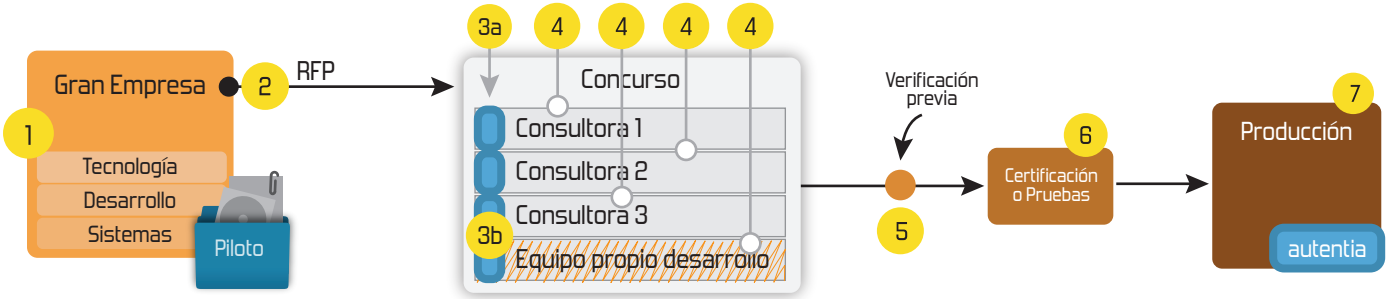
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

- Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery
- Gestor portales (Liferay)
- Gestor de contenidos (Alfresco)
- Aplicaciones híbridas
- Control de autenticación y acceso (Spring Security)
- UDDI
- Web Services
- Rest Services
- Social SSO
- SSO (Cas)
- JPA-Hibernate, MyBatis
- Motor de búsqueda empresarial (Solr)
- ETL (Talend)
- Dirección de Proyectos Informáticos.
- Metodologías ágiles
- Patrones de diseño
- TDD
- Tareas programadas (Quartz)
- Gestor documental (Alfresco)
- Inversión de control (Spring)
- BPM (jBPM o Bonita)
- Generación de informes (JasperReport)
- ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Samuel Martín Gómez-Calcerrada

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero en Informática, especialidad en Ingeniería del Software.

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/J2EE

Ver todos los tutoriales del autor

Fecha de publicación del tutorial: 2014-10-24

Liskov substitution

0. Índice de contenidos.

1. Introducción

El tercero de los principios S.O.L.I.D. corresponde al principio de sustitución de Liskov.
Fue propuesto por Barvara Liskov y Jeannette Marie Wing en la década de los noventa, aunque años antes, en el 1987, Liskov había hablado de él en una conferencia.

2. Liskov Substitution / Sustitución de Liskov

Fue definido formalmente así:

Let q(x) be a property provable about objects x of type T. Then q(y) should be provable for objects y of type S, where S is a subtype of T.

Que traducido a un lenguaje orientado a objetos sería similar a:

- Si a un método “q” le podemos pasar objetos “x” de la clase “T”, el método hace correctamente su función.
- Si tenemos una clase “S” que hereda de la clase “T”.
Entonces un objeto “y” de la clase “S” debería ser capaz de pasarse a la función “q” y ésta funcionará igualmente.

Hay una breve definición en la Wikipedia que me parece muy acertada.
Cada clase que hereda de otra puede usarse como su padre sin necesidad de conocer las diferencias entre ellas.

3. Ejemplo

Por poner un ejemplo de buen uso del principio de Liskov vamos a ver un ejemplo con vehiculos.

```
1  class InterfazVehiculo{
2      function acelerar();
3  }
4
5  class Camion{
```

```
    function acelerar() extends InterfazVehiculo{
        introducirMasCombustible();
    }
}

class CocheElectrico extends InterfazVehiculo{
    function acelerar(){
        incrementarVoltaje();
    }
}

class Conductor{
    function conducir(InterfazVehiculo vehiculo){
        // otras funcionalidades…
        v.acelerar();
    }
}
```

El conductor tiene un objeto **InterfazVehículo** cuya instancia pertenece a uno de sus hijos. Pero no necesita saber a cuál en concreto para hacerlo funcionar.
Seguir este principio hace mas sencilla la implementación de nuevas clases hijas.

4. Antiejemplo

Para terminar de entender este principio vamos a ver el ejemplo típico que lo viola.
En geometría, un cuadrado es un tipo de rectángulo particular que se distingue porque su base y su altura son iguales.
Vamos a modelar el diseño:

```
class Rectangulo{
    int base;
    int altura;
    //añadimos los getter y setter
    function area(){
        return base * altura;
    }
}

class Cuadrado extends Rectangulo{
    function setAltura(int altura){
        this.altura = altura;
        this.base = altura; // en un cuadrado ambos son iguales;
    }
    function setBase(int base){
        this.altura = base; // en un cuadrado ambos son iguales;
        this.base =base;
    }
}

class Usuario{
    function comprobarArea(Rectangulo rectangulo){
        rectangulo.setBase(4);
        rectangulo.setAltura(5);
        if (rectangulo.area() != 20){
            //entrar aquí implica que se viola el principio, un cuadrado tendría area
        }
        else{
            //buenaImplementacion
        }
    }
}
```

En este caso si la clase dinamica de Rectangulo es Cuadrado, el cálculo del área será erróneo, lo que nos forzaría a ensuciar y complicar el código con comprobaciones.

Este ejemplo me gusta por otra razón distinta a la del principio de Liskov, es que no se debe mapear automáticamente el mundo real en un modelo orientado a objetos. No existe una equivalencia unívoca entre ambos modelos.

Espero que os haya quedado claro, un saludo.

5. Principio S.O.L.I.D.

- Single Responsibility Principle / Principio de Responsabilidad Única
- Open-Closed Principle / Principio Abierto-Cerrado
- Liskov Substitution
- Interface Segregation Principle / Principio de segregación de interfaz
- Dependency inversion principle / Principio de inversión de dependencias

» Embeber vídeo en MailChimp

» Tutorial VIPER en Swift

Últimos Tutoriales del Autor

» [S.O.L.I.D.] Dependency inversion principle / Principio de inversión de dependencias

» [S.O.L.I.D.] Interface Segregation Principle / Principio de segregación de interfaz

» [S.O.L.I.D.] Open-Closed Principle / Principio Abierto-Cerrado

» [S.O.L.I.D.] Single responsibility principle / Principio de Responsabilidad Única

» Emmet.io - el toolkit esencial para los desarrolladores Web

Anímate y coméntanos lo que pienses sobre este **TUTORIAL:**

» **Registrate** y accede a esta y otras ventajas «

Fecha publicación: **2014-11-12-18:06:46**

Autor: **Crul**

Voy entendiendo, poco a poco. Para ciertos paradigmas (orientado a datos p.e.) además me encaja perfectamente, pero con otros no tanto. Según entiendo de DDD, sí es deseable que el modelo refleje el negocio, de manera que un analista de negocio pueda entierlo. Es ahí donde yo creo que me costaría explicar por qué "un cuadrado no es un rectángulo".

Comentándolo por aquí... creo que estoy llevando la idea de "modelo comprensible para la gente de negocio" demasiado lejos. Con un lenguaje ubicuo es suficiente, no creo que las herencias del modelo sean comprensible para negocio.

¡Más gracias!

Fecha publicación: **2014-11-12-14:40:33**

Autor: **smartin**

Muy buen ejemplo Crul.

Efectivamente parece contraintuitivo, pero porque partimos de la base de mapear el mundo real literalmente en lugar de utilizar un modelo que sea lógico para la funcionalidad que necesitamos.

Gracias por compartir el vídeo :)

Fecha publicación: **2014-11-12-12:34:36**

Autor: **Crul**

Voy a intentar contestarme a mí mismo...
De lo dicho por un compañero, y de lo que he entendido aquí:
https://www.youtube.com/watch?v=Orhu0x5aplI (a Tesla is NOT a Car)

Entonces... un Cuadrado NO es un Rectángulo, curioso Ó_ò

Fecha publicación: **2014-11-12-12:26:20**

Autor: **Crul**

Buenas,
Ante todo, gracias por los tutoriales. Este punto en concreto está siendo objeto de debate en la oficina. A ver si

El ejemplo de Rectángulo/Cuadrado siempre me ha vuelto loco porque, efectivamente veo los errores, pero no soy capaz de imaginar cuál sería la forma de solucionarlo.

Si eliminamos las líneas que diferencian Cuadrado de Rectángulo, entonces Cuadrado se convierte en una clase de marcado, y realmente no aporta ninguna funcionalidad respecto a la limitación "base=altura"... no me parece la "solución buena".

Otra posible salida es dejar los setter originales y añadir una comprobación en Cuadrado.area() que lance una excepción si base!=altura, pero en ese caso también se "rompe" el test y Cuadrado presenta un comportamiento distinto. Tampoco es una solución válida.

Otra que hemos comentado es darle la vuelta al modelo (Rectangulo extends Cuadrado) pero eso no es muy semántico... según lo entiendo yo.

¿Se os ocurre un ejemplo "bueno" del modelado de Rectángulo/Cuadrado que cumpla Liskov?
De nuevo, muchas gracias.