Nonfunctional Requirements

Cristian Camilo Serna Betancur

Andres Grisales Gonzalez

John Fredy Mejia Serna

September 19, 2020

Contents

Definición

Un requisito no funcional o atributo de calidad es un requisito que sabe bien y especifica criterios que pueden usarse para juzgar la operación de un sistema en lugar de sus comportamientos específicos.[1]

1 Performance

El rendimiento define qué tan rápido un sistema de software o su pieza en particular responde a las acciones de ciertos usuarios bajo cierta carga de

trabajo. En la mayoría de los casos, esta métrica explica cuánto debe esperar un usuario antes de que ocurra la operación destino.

2 Wrapability

Es la capacidad que tiene un sistema para ser *envuelto* por otro sistema, logrando de este modo, adjuntar nuevos comportamientos y proporcionando funcionalidades adicionales manteniendo intacto el sistema principal. Dicho de otro modo, agregar funcionalidad a un sistema existente sin alterar su estructura actual.

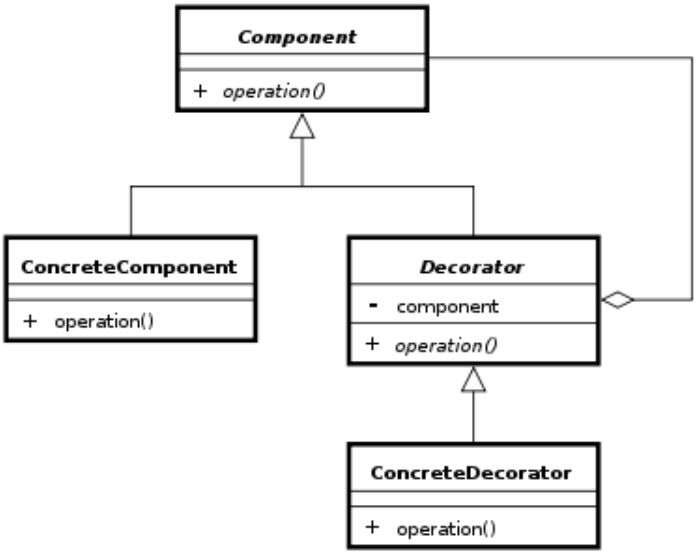

Figure 1: Decorator Pattern

3 Usability

La *usabilidad* es un requisito clásico no funcional[2] que responde a una simple pregunta: ¿Qué tan difícil es usar el producto?

3.1 Criterios de Usabilidad

Pueden existir muchos tipos de criterios de usabilidad[1], a continuacion presentamos algunos de ellos:

[1] Cabe destacar que cada persona podria tener sus propios criteros ya que no existe un concenso que lo especifique (como era de esperarce).

3.1.1 Capacidad de aprendizaje

Que tan rápido es para los usuarios completar las acciones principales una vez que ven la interfaz.

3.1.2 Memorabilidad

¿Pueden los usuarios volver a la interfaz después de un tiempo y comenzar a trabajar de manera eficiente con ella de inmediato?

4 Intercambiabilidad

Capacidad del software para ser usado en lugar de otro producto software, para el mismo proposito, en el mismo entorno. Es la habilidad de que un objeto puede ser reemplazado por otro sin afectar el código que usa el objeto. Esa posibilidad generalmente requiere que dos objetos compartan una interfaz que sea estrictamente igual o compatible en un caso particular. Usualmente se usa el patron fachada para lograr esta capacidad en el futuro.

5 Aciclicidad

Con la *aciclicidad* buscamos evitar dependencias cíclicas entre clases y entre paquetes, es una muy buena métrica para medir la mantenibilidad del software, cuando nos encontremos con dependencias cíclicas debemos sentarnos un rato y pensar como modificamos ya sea la ubicación de clases dentro paquetes de manera que podamos romper esa ciclicidad que es dañina.

5.1 Ejemplos

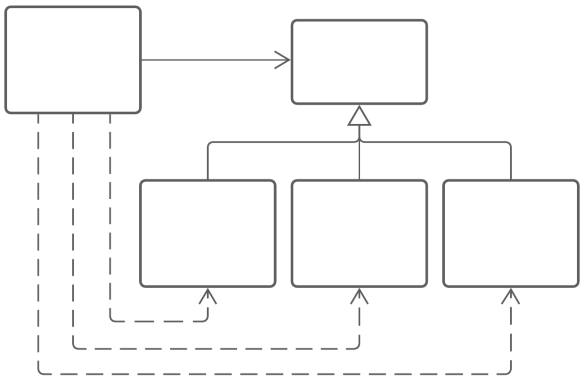

Figure 2: Esto no está bien.

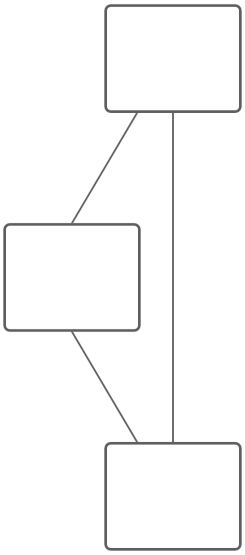

Figure 3: Esto tampoco está bien.

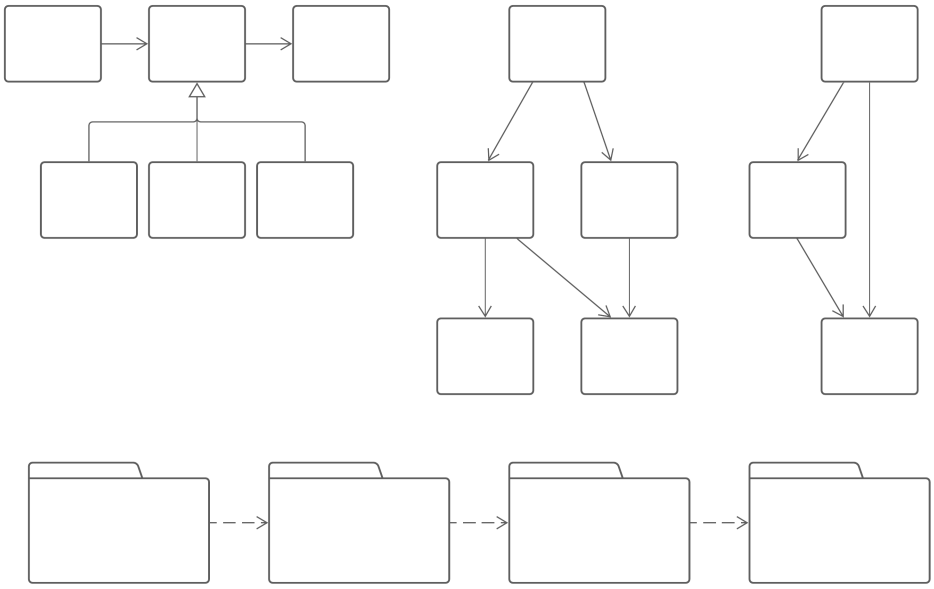

Figure 4: Esto está bastante bien.

6 Desacoplamiento

Cuando hablamos de *acoplamiento* nos referimos a la relación entre dos entidades en un sistema de software (generalmente clases). Cuando una clase usa otra clase, o se comunica con ella, se dice que "depende" de esa otra clase, por lo que estas clases están "acopladas". Al menos uno de ellos "sabe" sobre el otro. La idea es que deberíamos tratar de mantener el acoplamiento entre clases en nuestros sistemas lo más "flojo" posible: de ahí "acoplamiento suelto" o, a veces "desacoplamiento". De esta forma, una de las tantas maneras para evitar este acoplamiento, es el uso de interfaces.

7 Sustituibilidad

Buscamos la capacidad de *substitución* correcta de subtipos de una clase, la forma de tener una sustituibilidad adecuada es respetando correctamente el Principio de substitución de Liskov.

References

[1] Non-functional requirement - wikipedia. https://en.wikipedia.org/wiki/Non-functional_requirement.

[2] Non-functional requirements: Examples, types, approaches — altexsoft. https://www.altexsoft.com/blog/non-functional-requirements.