Verdadero o falso

Refactoring/testing:

* Cuando un código es refactorizado cambia su comportamiento agregando más funcionalidad.

**Falso, el refactoring no altera el comportamiento.**

* Si el codigo esta bien refactorizado no es necesario testear

**Falso, siempre es necesario testear para comprobar que no haya errores.**

* Después de ser refactorizado, la estructura interna del código permanece igual que antes

**Falso, la estructura interna del codigo cambia, lo que pertenece igual es su comportamiento.**

* La refactorización del código se hace en un solo paso en el que se unen todos los cambios

**Falso, la refactorizacion consiste de varios pasos para llevar a la solucion deseada.**

* El refactoring mejora el diseño del artefacto de software

**Verdadero.**

* Los refactorings (según Fowler) son estrategias que ayudan a eliminar errores frecuentes (bugs) del software

**Verdadero, parte de sus caracteristicas es ayudar a eliminar errores del software.**

* El refactoring no debe alterar la funcionalidad ofrecida por el artefacto de software al que se aplica

**Verdadero, no debe alterar la funcionalidad sino su estructura interna.**

* El refactoring es un proceso iterativo y continuo de mantenimiento de código

**No se. Puede que si, en el sentido de que cada tanto hay que aplicar refactoring para que no se deteriore el software.**

* Según Fowler, solo debemos invertir esfuerzo en refactoring cuando el código se vuelve ilegible

**Falso, es mejor no llegar al codigo ilegible. Cuando se detectan malos olores ya es una señal de que se necesita el refactoring.**

* Durante el refactoring los tests de unidad son necesarios para asegurar que no se altera el comportamiento del artefacto de software  
  **Verdadero, tienen esa utilidad.**
* Si uso los refactorings del System Browser de pharo, los tests se actualizan automáticamente  
  **???**
* Si hago refactoring con mucho cuidado, no es necesario que programe ni ejecute tests  
  **Falso, es necesario el uso de test para asegurar que no se altero la funcionalidad.**
* Un refactoring bien hecho nunca rompe los tests de unidad  
  **Falso, el refactoring puede estar bien hecho pero puede haber test coupling?**
* Puede ocurrir que un refactoring rompa los tests de unidad aunque sea correcto  
  **Si, por acoplamiento de los test.**
* Mock objects, test dummies, test stubs y test spies son todos test doubles

**Falso, falta el Fake Object que imita al original en todo lo posible, hasta los tiempos de respuesta.**

**Es verdadero, son todos test doubles, yo lei “son todos los test doubles”.**

* Cuando un test falla retorna un test dummy que es polimórfico con TestResult

**No se**

* Se usan test doubles cuando alguno de los objetos requeridos no está disponible en el ambiente de test  
  **Verdadero, si esta es su funcionalidad. Puede ser porque no este desarrollado, sea dificil de usar o tarda mucho en darnos el estado que requerimos a testear.**
* Mock objects, test doubles, test stubs y test spies son todos test dummies  
  **Falso, mock objects, test stubs, test spies y test dummies son todos test doubles.**
* El refactoring replace temp with query puede aplicarse para eliminar el mal olor ‘método largo’  
  **Si, es una manera de solucionar el metodo largo.**
* Un método es largo si tiene más de 6 lineas  
  **Falso, si tiene mas de 10 ?**
* El refactoring replace extract method puede aplicarse para eliminar el mal olor ‘método largo’  
  **Falso, no hay replace extract method, si extract method, ese ayudaria.**
* Comment every important line puede aplicarse para eliminar el mal olor ‘método largo’  
  **Falso, esto no ayudaria a eliminar el mal olor metodo largo, de hecho lo aumentaria.**
* Un método largo indica la necesidad de hacer un refactoring

**Verdadero, es un mal olor.**

Frameworks:

* En un framework de caja blanca, los hotspots pueden verse con mayor claridad que en un framework de caja negra y por eso se llama así

**Falso, los frameworks de caja blanca son los que utilizan herencia en sus puntos de extension. Ademas, los hotspot se pueden ver tanto en caja blanca como caja negra.**

* En un framework de caja negra se utilizan la configuración y composición de objetos como mecanismos principales de extensión  
  **Verdadero.**
* En un framework de caja negra los frozen spots están ocultos y en un framework de caja blanca están visibles y pueden modificarse.  
  **Falso, no se pueden modificar ningun frozenspot ya sea caja blanca o negra, ademas, ademas pueden verse los frozenspots de frameworks de caja negra, no hay ninguna regla que lo prohiba, este concepto esta relacionado a cual es su metodologia para extender sus puntos de extension y que tanto se debe saber del funcionamiento interno del framework para crear hotspots.**
* En un framework de caja blanca se utiliza herencia como mecanismo principal de extensión  
  **Verdadero.**
* Se llaman frameworks de caja blanca a las librerías de clase y frameworks de caja negra a los editores gráficos de aplicaciones

**Falso. Las librerias de clase no son un framework, es la aplicación la que maneja a la libreria, hay una gran diferencia entre ellas.**

* En un framework orientado a objetos, los frozen spots son siempre clases abstractas y los hotspots son siempre clases concretas

**Falso. No todas las clases de los frozen spots son clases abstractas.**

* Si un programa solo tiene hotspots decimos que es un framework y si tiene solo frozenspots decimos que es una librería  
  **Falso, un framework tambien tiene frozen spot.**
* Todas las aplicaciones construidas con un framework son iguales en relación al comportamiento que el framework captura en sus frozenspots  
  **Verdadero, es asi como funciona.**
* Los hotspots de un framework permiten que el mismo pueda utilizarse para construir aplicaciones diferentes

**Verdadero, es este el objetivo.**

* Todo framework bien diseñado debe tener la misma cantidad de frozenspots que hotspots

**Falso, esto no tiene nada que ver, y es posible que nunca tengan la misma cantidad.**

Multiple choice

Patrones:

* Seleccionar las situaciones que son indicadores de aplicabilidad del patron decorator:
  + **Para agregar responsabilidades? a los objetos de forma dinámica y transparente, es decir, sin afectar a otros objetos**
  + Cuando un comportamiento en común entre varias subclases debe ser factorizado y ubicado en una superclase común para evitar duplicación de código
  + Para mantener bajo control las ?? que pueden hacer las subclases
* Seleccionar las situaciones que son indicadores de aplicabilidad del patrón composite:
  + Definir el esqueleto de un algoritmo en una operación, delegando algunos pasos a las subclases.
  + Proporcionar un intermediario a un objeto para controlar su acceso.
  + Agregar comportamiento a un objeto en forma dinámica y transparente
  + Convertir la interfaz de una clase a otra interfaz que el cliente espera
  + **Componer elementos en estructuras de árbol para representar jerarquías parte-todo**
  + **Los clientes tratan uniformemente a objetos individuales y a composiciones**

Preguntas escritas

* ¿Qué significa la expresión “código con mal olor”?

Significa que dentro del mismo existen patrones que indican que en el futuro podria traer problemas a la aplicación, ya sea de eficiencia, adaptabilidad, mantenimiento, bugs o agregar nueva funcionalidad.

* Indique 3 razones por las cuales es importante realizar un refactoring.

1- Es importante dado que ofrece una ayuda a futuro para poder detectar bugs.

2- Mejora el diseño del codigo del software.

3- Permite que se agregue funcionalidad de forma mas agil.