8 PRINCIPLES OF BETTER UNIT TESTING

Para escribir buenas pruebas unitarias, se deben seguir estas pautas:

1. **Sepa lo que está probando:** Una prueba escrita sin un objetivo claro es fácil de detectar (larga, difícil de entender, evalúa mas de una cosa). Un truco es utilizar el escenario probado y el resutado esperado como parte del nombre del método de prueba. Probar solo una cosa hace que sea más legible y es más facil encontrar la causa si falla.

2. **Las pruebas unitarias deben ser autosuficientes:** Una buena prueba unitaria debe estar aislada. Una sola prueba no debería depender de ejecutar otras pruebas antes ni del orden de ejecución. Ejecutarla 1000 veces debería dar siempre el mismo resultado. El uso de estados globales pueden causar "fugas" entre las pruebas.

3. **Las pruebas deben ser deterministas:** Una prueba debe pasar todo el tiempo o fallar hasta que se solucione. Otra "práctica" que debe evitarse es escribir pruebas con entrada aleatoria ya que introduce incertidumbre.

4. **Convenciones de nomenclatura:** Lo primero que nota sobre una prueba fallida es su nombre. Cuando falla una prueba bien nombrada es más fácil entender qué se probó y por qué falló. Por ejemplo para una clase Calculadora que puede dividir dos números un buen nombre de test es Dividir_DividirPorCero_LanzaExcepcion().

5. **Repítete:** En las pruebas unitarias es muy importante la legibilidad por lo que es aceptable el código duplicado.

6. **Resultados de las pruebas, no implementación:** Las pruebas unitarias exitosas solo fallarían en caso de un error real o un cambio de requisitos. Evitar escribir pruebas frágiles, ya que fallarían debido a cambios internos en el software que no afectan al usuario. Solo pruebe métodos privados si tiene una muy buena razón. La refactorización trivial puede causar errores complicados y fallas en las pruebas.

7. **Evitar especificaciones excesivas:** Use un marco de aislamiento para establecer el comportamiento predeterminado de los objetos externos y asegurarse de que no esté configurado para lanzar excepción si se llamó a un método inesperado.

8. **Utilice un marco de aislamiento:** Un marco de aislamiento es una biblioteca de terceros y un gran ahorro de tiempo. Los ahorros en líneas de código entre el uso de un marco de aislamiento y la escritura de simulaciones a mano para el mismo código pueden llegar al 90%. Cada marco tiene un conjunto de API para crear y usar objetos falsos sin que el usuario necesite mantener detalles irrelevantes de la prueba específica. Si se crea una falsificación para una clase específica, cuando esa clase agrega un nuevo método, no es necesario cambiar nada en la prueba.