**Clean Code**

**A Handbook of Agile**

**Software Craftsmanship**

"In short, a programmer who creates clean code is an artist who can transform a blank canvas into an elegant code system." (Martin, 2009)

**CHAPTER 8**

Boundaries

El código a terceros nos permite tener mayor funcionalidad en menos tiempo y es importante realizar las pruebas necesarias para que nosotros logremos entender el código de terceros, denominadas pruebas de aprendizaje. Estas pruebas son gratuitas y rentables, lo que nos brinda mayor manejo y accesibilidad para realizar las pruebas. Estas pruebas de aprendizaje nos indican que el paquete de terceros que estamos usando funciona de la forma esperada. Si el paquete de terceros cambia, lo sabremos en ese instante ya que las pruebas antes realizadas mostraran que ya no se realiza la función deseada.

Es mejor controlar y manejar algo que ya conocemos a algo del cual no dependemos.

**CHAPTER 9**

Unit Tests

Leyes para crear pruebas de unidad o desarrollo guiado por las pruebas (GDP):

1.- No debe crear código de producción hasta que haya creado una prueba de unidad que falle.

2.- no debe crear más de una prueba de unidad que baste como fallida y no compilar se considere un fallo.

3.- no debe crear más código de producción que el necesario para superar la prueba de fallo actual.

Las pruebas y e l código se crean de forma conjunta, tomando en cuenta que primero se analizan las pruebas y después el código. Así se generaran varias pruebas que ayudaran a que nuestro programa sea mejor y más eficiente. Mientras más pruebas se realizan al código, nuestro programa será de mejor calidad.

Las pruebas deben cambiar de acuerdo a la evolución del programa, por eso e s importante que las pruebas sean actualizadas y limpias. Las pruebas ayudan a que el código sea flexible y se pueda mantener y reutilizar. Sin pruebas limpias, se provoca un estancamiento en el código y fallas ya que estas pruebas no se las puede modificar en relación con el código actual.

El elemento más importante en una prueba de código es la LEGIBILIDAD.

FIRST:

Son los 5 elementos que debe seguir una prueba limpia:

**Rápido**: las pruebas deben ser rápidas ya que si no lo son no detectan los problemas.

**Independencia**: debe poder ejecutar cada prueba de forma independiente y en el orden que desee. Si dependen de otra, cuando una falle se generara una fila de fallos.

**Repetición**: las pruebas pueden ejecutarse en cualquier entorno.

**Validación** **automática**: deben ser de tipo booleanos, si pasa verdadero o sino falso.

**Puntualidad**: las pruebas deben crearse en el momento precios, antes del código.

**CHAPTER 10**

Classes

Se habla sobre las accesibilidades que pueden tener nuestros atributos y métodos en una clase, ya sean privados, públicos o protegidos. El encapsulamiento nos sirve para ocultar cierta información. Las clases deben ser de tamaño reducido. El nombre de la clase debe especificar su funcionalidad. Mientras más variables manipulen un método más cohesión tendrá una clase, y eso es lo que se busca. Lograr una cohesión nos muestra que cada método y variable están relacionados entre sí y tiene un orden lógico. Si se realizan cualquier tipo de cambio es probable que cambie todo el código es por eso que las clases deben tener un orden y así cambiar el código será más sencillo. Existen las clases concretas donde son la base de nuestro código y las clases abstractas que representan conceptos.

**CHAPTER 11**

Systems

La separación de aspectos es una de las técnicas más importantes que se deben aplicar ya que estas nos ayudan a tener un código más limpio y seguir un orden. El código de proceso de inicio no debe confundirse con la ejecución. La función main crea los objetos necesarios para el sistema la pasa a otra función y esta se encarga de realizar toda la ejecución del programa. Un buen programador debe tener en cuenta la arquitectura que debe tener su programa, ya que esta ofrece agilidad. En todos los niveles de abstracción los objetos debes ser claros.

**CHAPTER 12**

Emergence

Un diseño es sencillo si cumple estas cuatro reglas:

Ejecuta todas las pruebas: un sistema que cumpla con todas las pruebas se lo denomina sistema testable. Esto provoca que diseños clases reducidas. Si nuestro sistema se puede probar podemos crear mejores diseños. La creación de pruebas nos permite tener un mejor diseño de nuestro programa.

No contiene duplicados o eliminar duplicados: existe la posibilidad que tengas varias líneas que cumplan la misma función que otra y simplemente están empeorando nuestro código, es importante definir que hace cada línea y verificar que no existan ninguna parecida.

Expresa la intención del programador: el código debe ser claro y entendible para el programador y para el resto de los programadores, si bien es cierto que cuando uno está programando lo hace entendible para si mismo pero hay que tener en cuenta que ese código puede ser de utilidad para otro programador y necesita entender cuál es el funcionamiento.

Refactorizar: al aumentar líneas en el código es importante tomarnos el tiempo necesario para identificar si nuestro diseño tiene relación, si tiene algún aspecto inconforme lo limpiamos y lo volvemos a ejecutar con las pruebas