Tu socio tecnológico

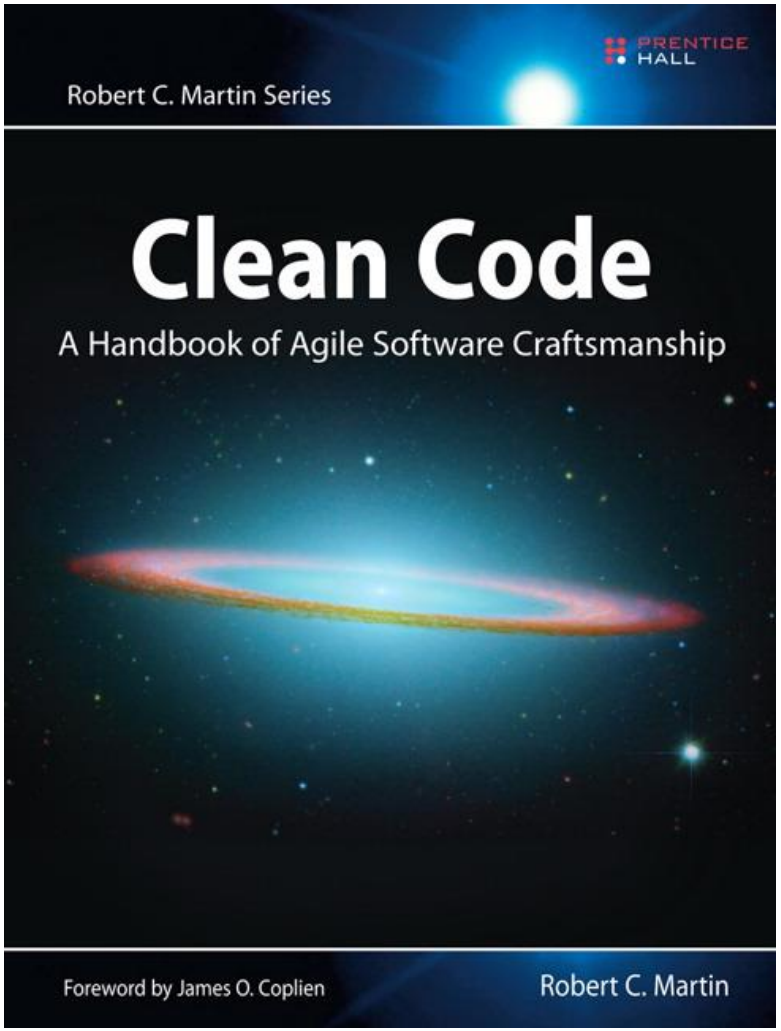
nunsys®

COMUNICACIONES · SISTEMAS · SOFTWARE · MARKETING · FORMACIÓN

# Clean code



28 de septiembre 2016



# Nombres con sentido

- Usar nombres que revelen las intenciones
- Evitar la desinformación
- Realizar distinciones con sentido
- Usar nombres que se puedan pronunciar
- Usar nombres que se puedan buscar
- Evitar codificaciones
- Nombres de clases con nombres descriptivos
- Una palabra por concepto, evitar utilizar fetch, retrieve y get como nombres equivalentes
- Evita usar la misma palabra con dos fines distintos.
- Usar nombres de dominios reconocidos, por ejemplo, patrones de diseño.
- Usar nombres del dominio del problema, lenguaje ubicuo.

# Funciones

- ¡Pequeñas!
- Hacer una cosa
- Un nivel de abstracción por función
- La regla descendente: leer el código de arriba abajo
- Evitar sentencias de selección múltiple
- Parámetros de funciones, 2 o menos parámetros
- Parámetros de indicador (Parámetros de tipo boolean)
- Objetos como parámetros
- Verbos y palabras clave
- Sin efectos laterales
- Evitar los parámetros de salida
- Separar las acciones de las consultas (Command Query Separation)
- Mejor lanzar excepciones que códigos de error
- Separar el procesamiento de errores
- DRY Principle (Don’t Repeat Yourself)

# Comentarios

- No hay nada más útil que un comentario bien colocado. No hay nada que colapse más un módulo que comentarios dogmáticos innecesarios. No hay nada más dañino que un comentario antiguo que propague mentiras y desinformación. Hacer una cosa
- Los comentarios siempre son fallos, los utilizamos porque no siempre sabemos cómo expresarnos sin ellos pero su uso no es motivo de celebración
- Comentarios de calidad:
  - Comentarios legales.
  - Comentarios de advertencia.
  - Comentarios TODO
  - Comentarios que se utilizan para amplificar la importancia de algo.
  - Javadoc en API públicas
- Sentencias de selección múltiple
- No use comentarios si puede usar un nombre de clase, nombre de función o nombre de variable adecuados
- Siempre que tenga la necesidad de introducir un comentario piense si no hay otra forma de expresarlo en el código

# Clases

- Las clases deben ser de tamaño reducido
- Aplicar principios SOLID
- Separar la construcción de una objeto de su uso
  - Uso de factorías
- Ley de Demeter. Un método de una clase solo puede llamar a métodos de:
  - La propia clase
  - Un argumento del método
  - Cualquier objeto creado dentro del método
  - Cualquier propiedad/campo directo de la propia clase

  Se pretende reducir el acoplamiento de clases y lo que se denomina choque de trenes:

  objetoA..metodoDeA().metodoDeB().metodoDeC();

# Procesamiento de errores

- Usar excepciones  en vez de funciones que devuelven código de error.
- Redacta mensajes de error informativos y pásalos junto a la excepción.
- Evitar pasar null en parámetros de  funciones siempre que sea posible.
- Evitar dejar excepciones sin gestionar

```
Try {
…
} catch (Exception ex) {}
```

# Procesamiento de errores. No devolver null

```
Ilist<Empleado> ObtenerEmpleadosDepartamento(String departamentoId) {
    IList<Empleado> listaEmpleadosDelDepartamento;

    if (listaEmpleados.count > 0) {
       listaEmpleadosDelDepartamento = (listaEmpleados.Where<Empleado>(
                  empleado => empleado.DepartamentoId = departamentoId).ToList<Elemento>();
    }

    return listaEmpleadosDelDepartamento;

}
```

**Mal**

```
...

Ilist<Empleado> empleados = ObtenerEmpleadosDepartamento(departamentoId);

If (empleados != null) {
     ...

}
```

```
Ilist<Empleado> ObtenerEmpleadosDepartamento(String departamentoId) {
    IList<Empleado> listaEmpleadosDelDepartamento = new List<Empleado>();

    if (listaEmpleados.count > 0) {
       listaEmpleadosDelDepartamento = (listaEmpleados.Where<Empleado>(
                  empleado => empleado.DepartamentoId = departamentoId).ToList<Elemento>();
    }

    return listaEmpleadosDelDepartamento;

}
```

**Bien**