FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Diseño de Código

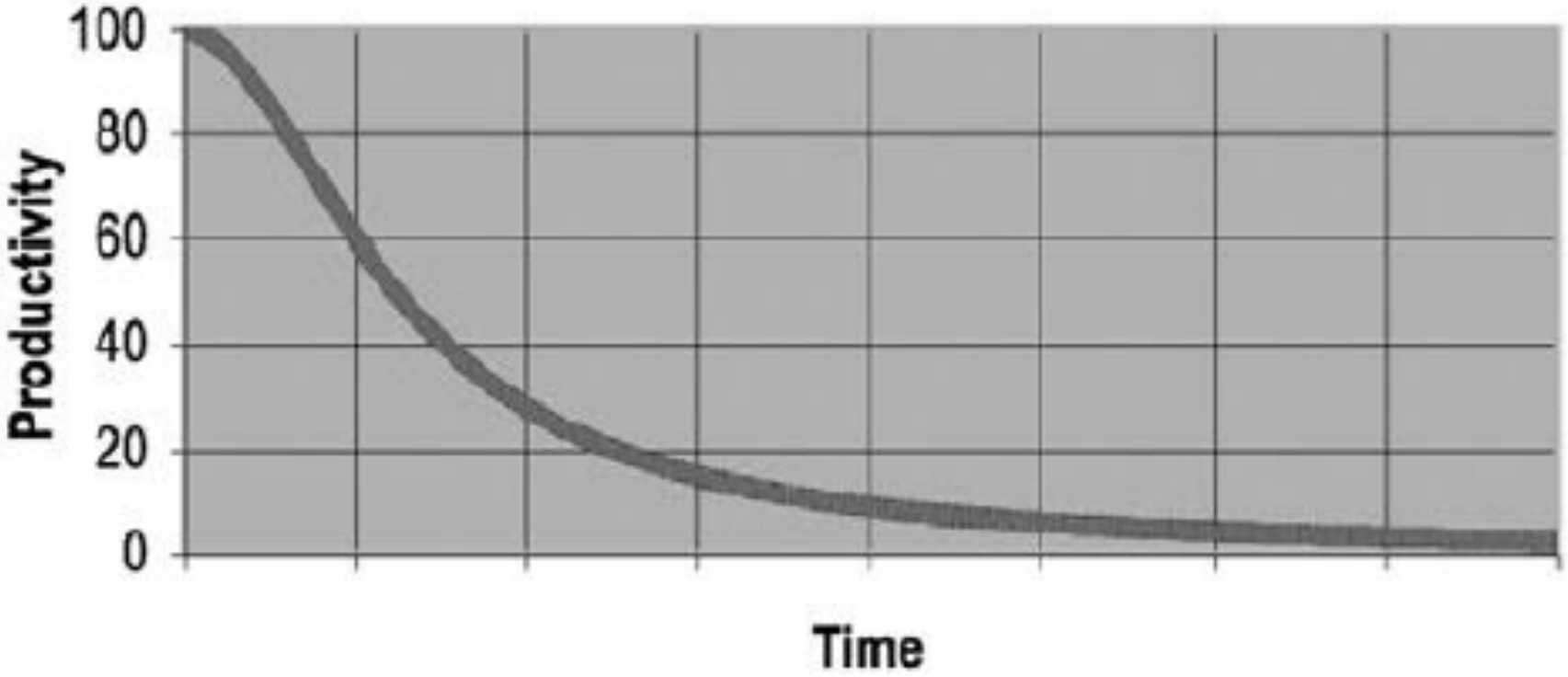
FI.UBA

75.10 - Técnicas de Diseño

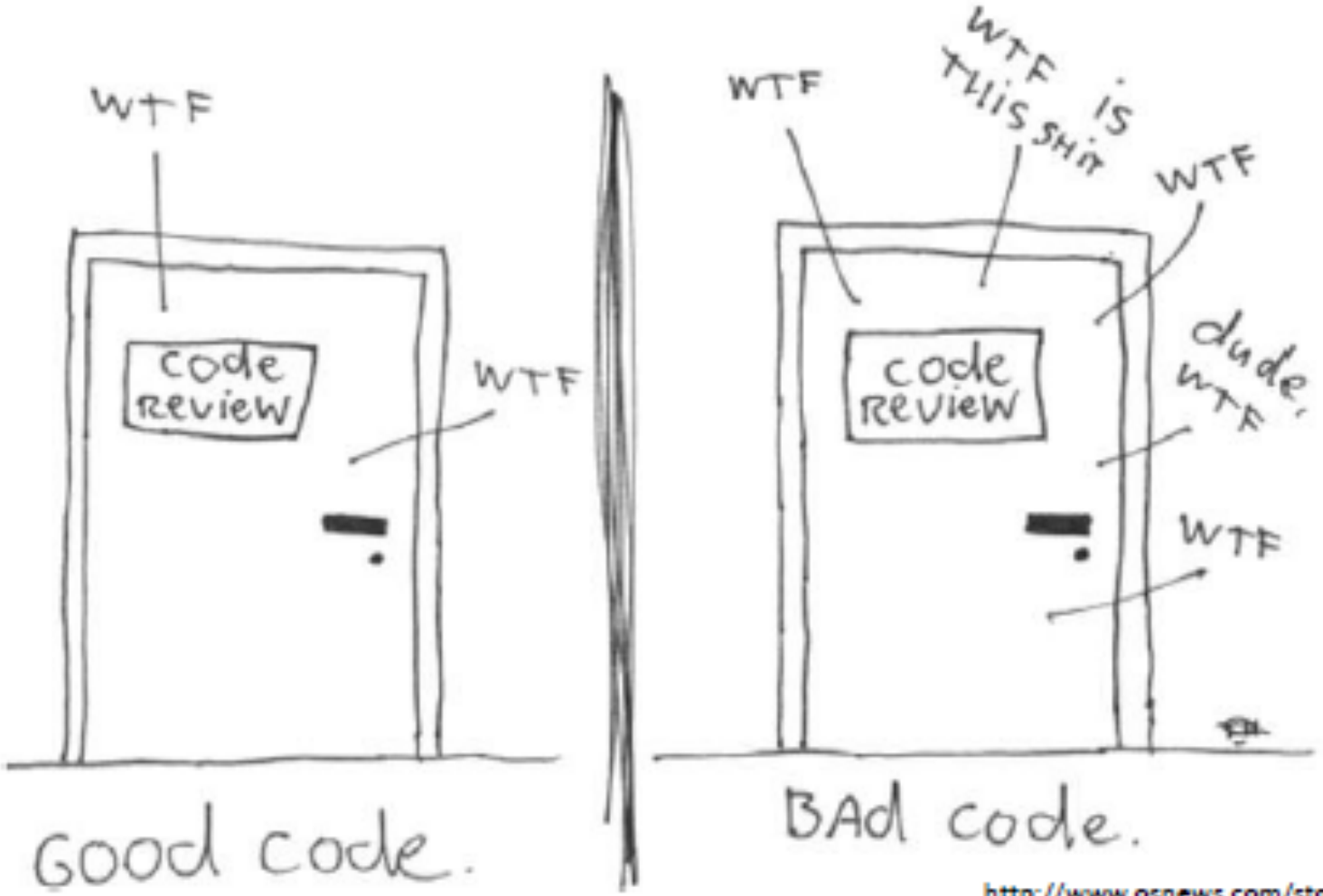
ORGULLO

PROFESIONALISMO

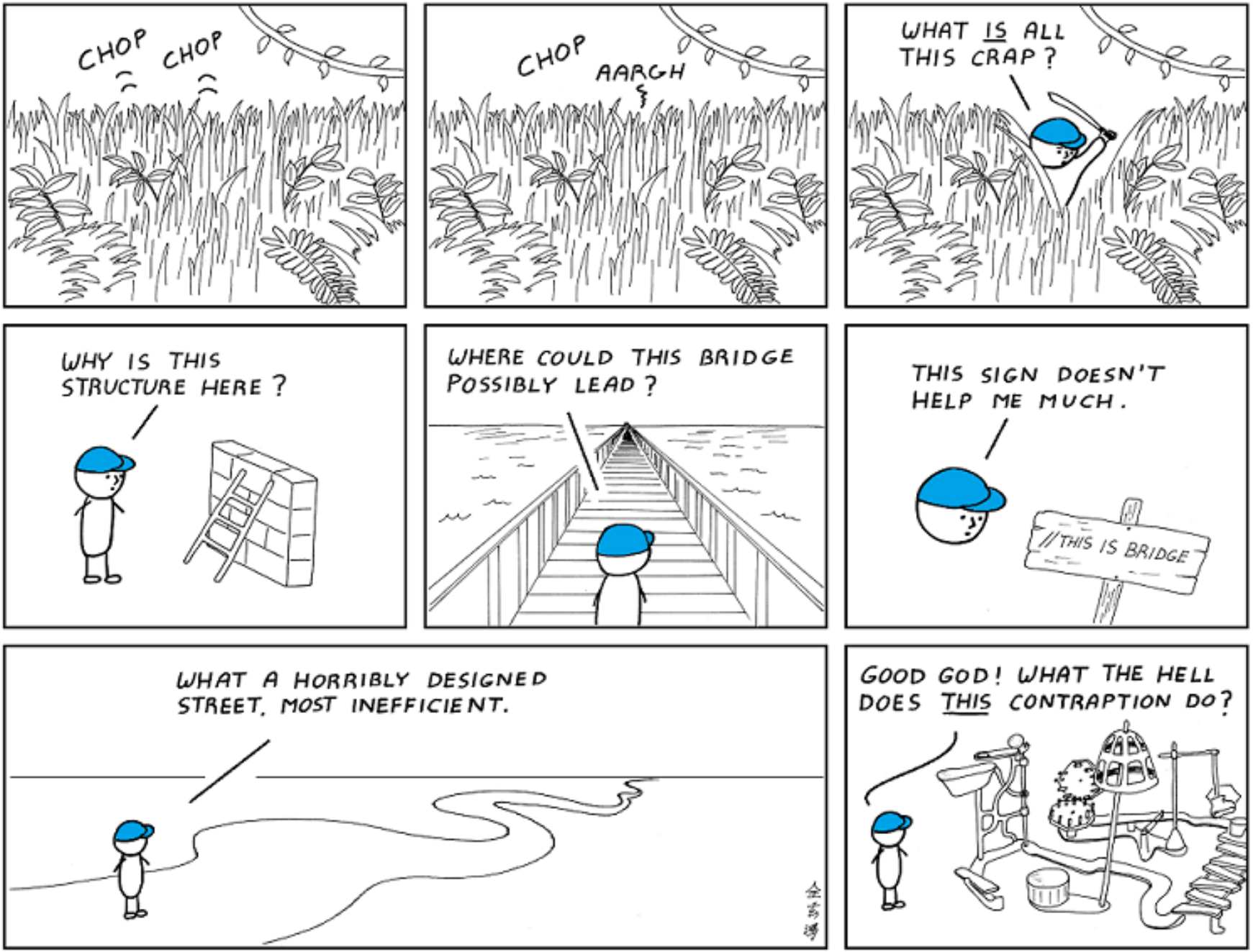
Costo de Poseer código no mantenible



The ONLY VALID MEASUREMENT OF Code QUALITY: WTFs/MINUTE



http://www.osnews.com/story/19266/WTFs



I hate reading other people's code.

Objetivos

- Mantenibilidad
- Simplicidad
- Claridad
- Flexibilidad
- Legibilidad

Nombres Significativos

```
int d; // Tiempo transcurrido en días
```

Preferir Nombres claros a cometarios

```
int tiempoTranscurridoEnDias;
int diasTranscurridosDesdeCreacion;
```

Usar nombres que revelen su intención

```
public List<int[]> obtener()
{
   List<int[]> lista1 = new
     ArrayList<int[]>();
   for (int[] x : laLista)
       if (x[0] == 4)
                lista1.add(x);
   return lista1;
}
```

- 1.¿Qué tipos de cosas se almacenan en la Lista?
- 2.¿Cual es el significado del item “CERO”?
- 3.¿Cuál es el significado del valor 4?
- 4.¿Para que se utiliza la lista que retorna ese método?

Usar nombres que revelen su intención

```
public List<Celda> obtenerCeldasConBanderas()
{
    List<Celda> celdasConBanderas = new ArrayList<Cell>();
    foreach (Celda celda in tableroJuego)
       if (celda.estaConBandera())
         celdasConBanderas.add(celda);

    return celdasConBanderas;
}
```

Use Searchable Names

- No dejar valores fijos, usar constantes con nombre claros.

```
int s = 0;
for (int j=0; j<34; j++) {
  s += (t[j]*4)/5;
}
```

```
int realDaysPerIdealDay = 4;
const int WORK_DAYS_PER_WEEK = 5;
int sum = 0;
for (int j=0; j < NUMBER_OF_TASKS; j++) {
   int realTaskDays = taskEstimate[j] * realDaysPerIdealDay;
   int realTaskWeeks = (realdays / WORK_DAYS_PER_WEEK);
   sum += realTaskWeeks;
}
```

Use nombres pronunciables

```
class DtaRcrd102
{
    private date credmahms;
    private date moddmahms;
    private string pszqint = "102";
};
class Cliente
{
    private date fechaCreacion;
    private date fechaModificacion;
    private string clienteId = "102";
}
```

Notaciones

Member Prefixes

```
public class Part {
   private String m_dsc; // The textual description
   void setName(String name) {
       m_dsc = name;
   }
}
```

```
public class Part {
   String description;
   void setDescription(String description) {
       this.description = description;
   }
}
```

Class Names

Method Names

Pick One Word per Concept

fetch, retrieve, get

Use Solution Domain Names

Funciones

- **Funciones pequeñas**

Deberían tener menos de 15 líneas aprox. por función

- **Hacer una sola cosa**

Single Responsibility Principle

- **Un solo nivel de abstracción por función**

Identificar distintos niveles de abstracción

Es la clave para reducir el tamaño de funciones y hacer una sola cosa por función

- **Leer de Arriba hacia abajo**

Como un periódico

- **Switch**

Evitarlo, rompe la regla de solamente una cosa

- **Argumentos**

Uno es bueno, Cero es mejor

EscribirArchivoEnDisco(archivo)

- **Flag**

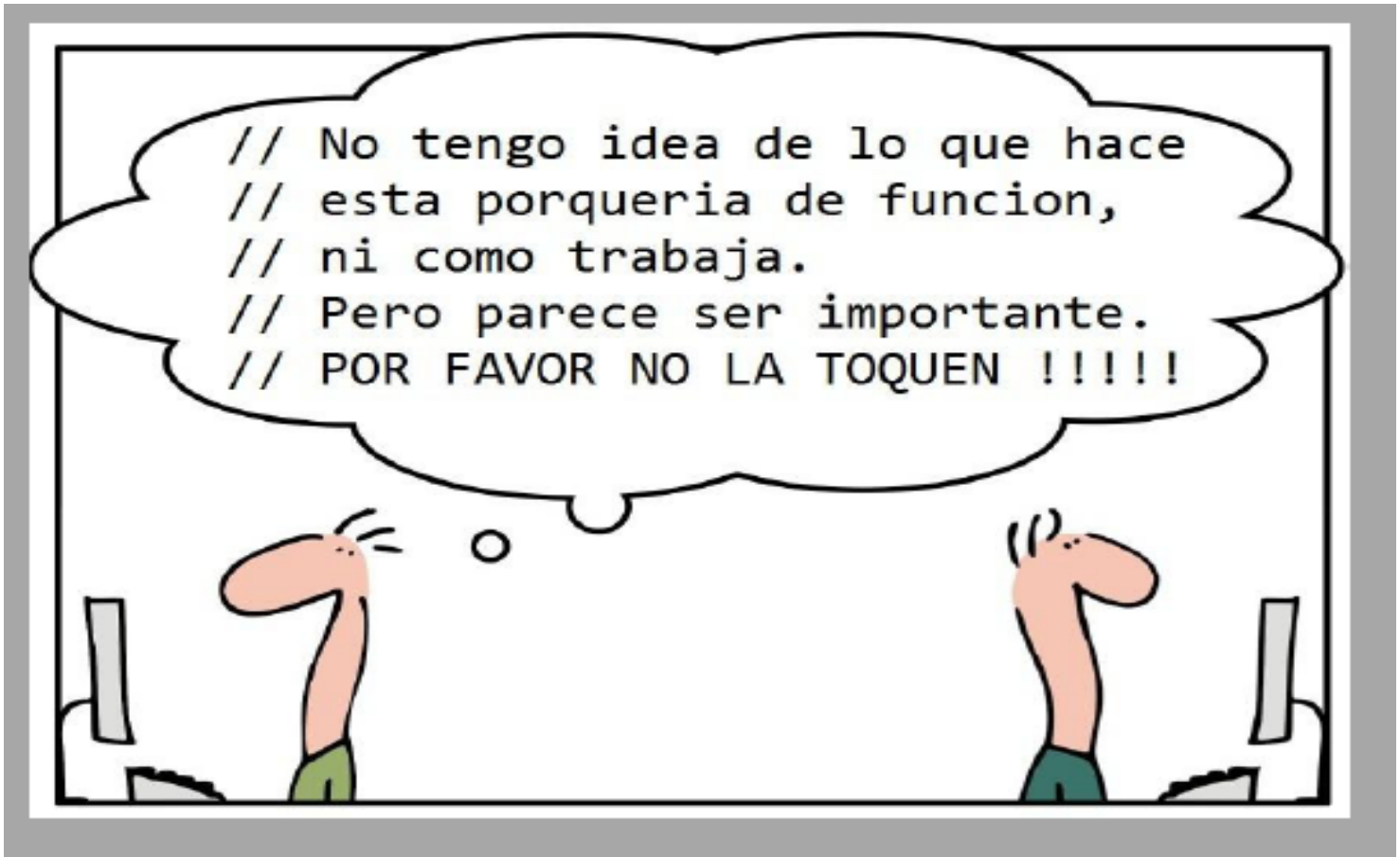
```
Mostrar(true)
```

Es preferible usar polimorfismo, o crear nuevas funciones

```
mostrarEnDesarrollo()
mostrarEnPruebas()
```

Comentarios

- Un código bien escrito no debería requerir comentarios



- **Explíca todo en código**

```
// Verifica si el empleado es candidato a
// obtener beneficios sociales
if ((empleado.tipoEmpleado =
  EMPLEADO_PLANILLAS) && (empleado.edad >
  65))
```

```
if
  (empleado.esCandidatoBeneficiosSociales())
```

Buenos Comentarios

- **Legal:**

```
// Derechos reservados por Seriva Inc. 2012
// Lanzado bajo GNU General Public License version 2.
```

- **Informativos:**

```
// format matched kk:mm:ss EEE, MMM dd, yyyy
Pattern patronTiempo = Pattern.compile( "\\d*:\
  \d*:\\d* \\w*, \\w* \\d*, \\d*");
```

- **Explicación de una intención:**

```
// Este es nuestro mejor intento de obtener una
// condición de un gran numero de hilos.
```

Buenos Comentarios

- **Clarificación:**

```
assertTrue(a.compareTo(a) == 0); // a == a
assertTrue(a.compareTo(b) != 0); // a != b
```

- **Advertencias de consecuencias:**

```
// No correr este test a menos que
// tengas bastante tiempo (Demora).
```

- **TODO:**

```
// TODO: Tarea a realizar
```

- **Ampliar información:**

```
// Indica más información relevante
```

Malos Comentarios

- **Redundancia:**

```
// Declaro una variable "x" del tipo entero
int x;
```

- **Comentario erróneo:**

Puede introducir errores

- **Comentarios obligatorios:**

Tienden a que se utilicen de manera inadecuada

- **Comentarios tipo Diario:**

Existen repositorios de código fuente para hacer esta tarea

Malos Comentarios

- **Ruido:**

```
/* Constructor por defecto */
protected AnnualDateRule()
```

- **Marcadores de posición:** `/**********`

- **Al cerrar una llave:**

```
} // if, Si las funciones son cortas no es necesario
```

- **Código comentado (muerto):**

```
// if (prueba == true) { }
```

Formato

```
package fitnesse.wikitext.widgets; import
java.util.regex.*; public class BoldWidget
extends ParentWidget { public static final
String REGEXP = "'''.+?'''"; private static
final Pattern pattern = Pattern.compile("'''(.
+?)'''", Pattern.MULTILINE + Pattern.DOTALL);
public BoldWidget(ParentWidget parent, String
text) throws Exception { super(parent); Matcher
match = pattern.matcher(text); match.find();
addChildWidgets(match.group(1));} public String
render() throws Exception { StringBuffer html =
new StringBuffer("<b>");
html.append(childHtml()).append("</b>"); return
html.toString();
```

Excepciones

- **Usar excepciones en vez de códigos de error**

  If (deletePage(page) == E_OK) { ….

- **En general no retornar Null**

- No hay nada más importante que escribir código de calidad para el éxito de un proyecto.

- Leer código debería ser como leer una novela – Grady Booch

- **Cualquier tonto** puede **escribir código** que una computadora puede comprender. **Buenos programadores escriben código** que otros humanos pueden entender.

  – Martin Fowler, 2008.

Herramientas útiles

- - StyleCop
- - FxCop
- - Eclipse
- - Check Style

Bibliografía

- Clean Code (Robert C. Martin)
- Code Complete, Steve McConnel (obligatorio, clásico, nivel inicial)
- Implementation Patterns, Kent Beck (nivel intermedio)
- Refactoring, Martin Fowler (nivel intermedio, clásico)