

# Diseño de Código

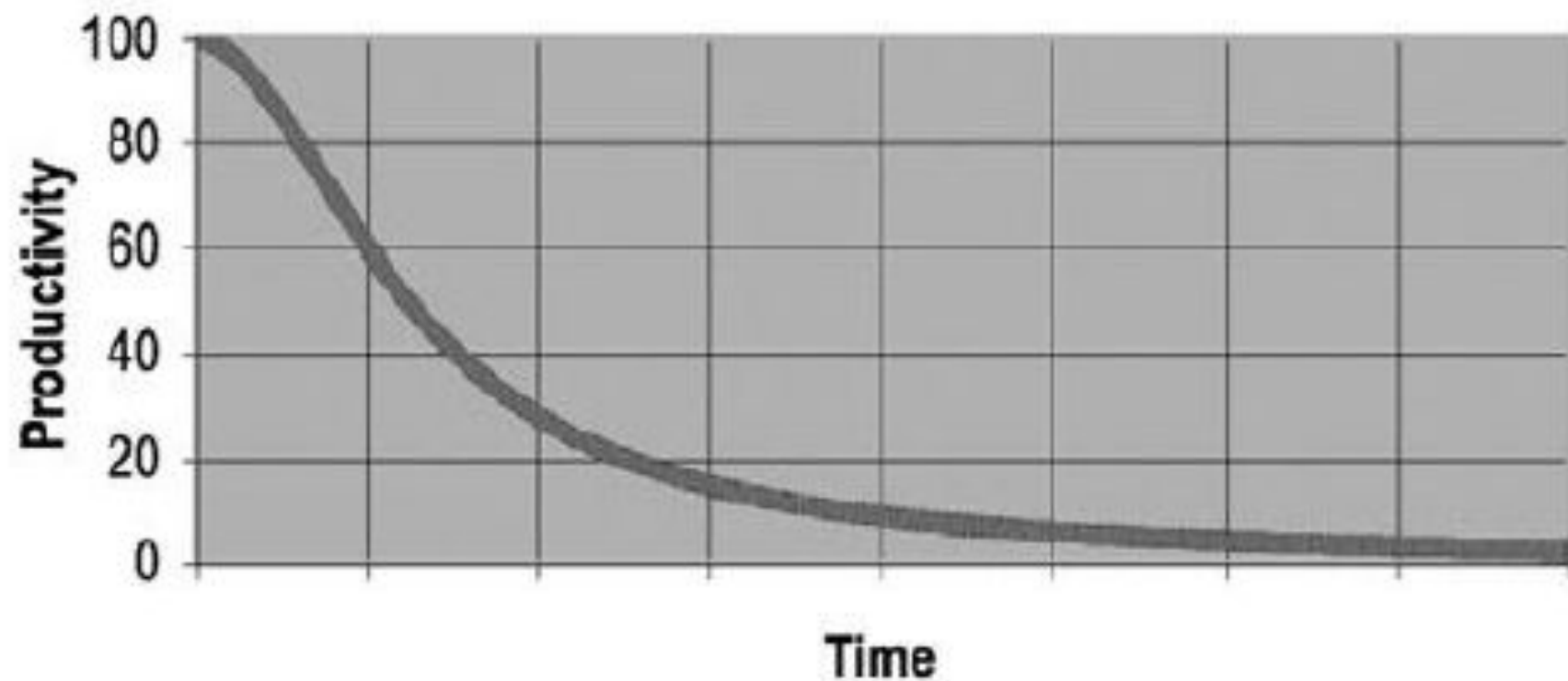
FI.UBA

Ingeniería de Software

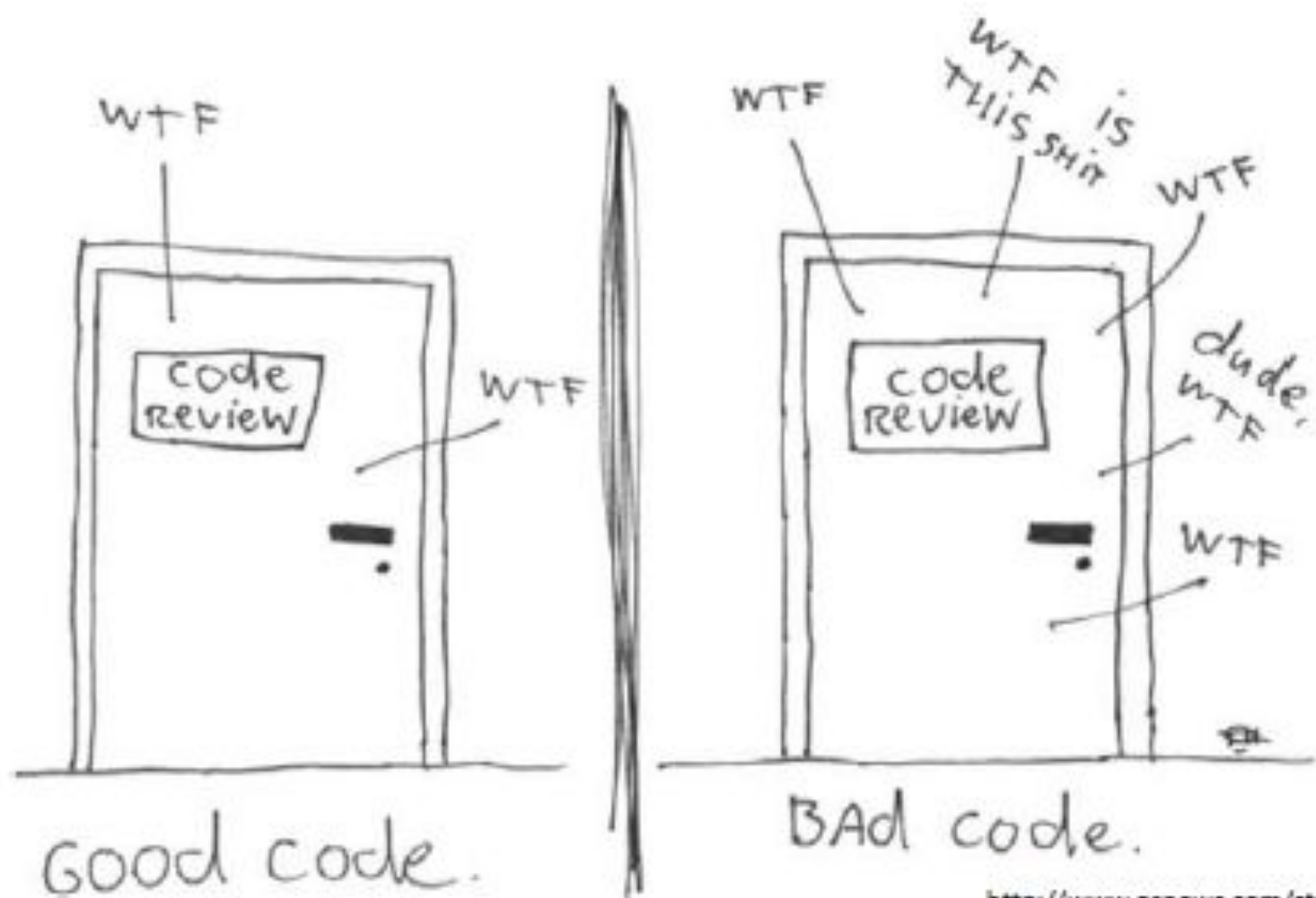
ORGULLO

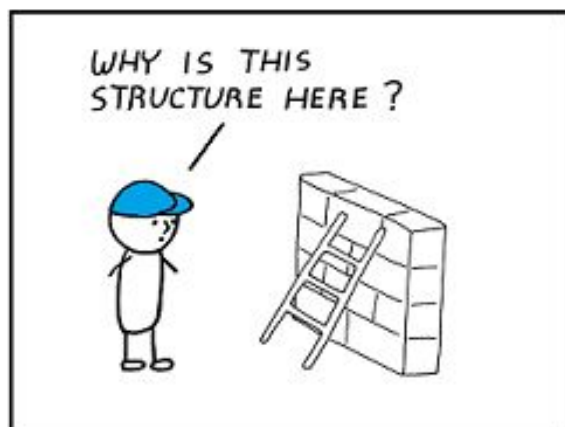
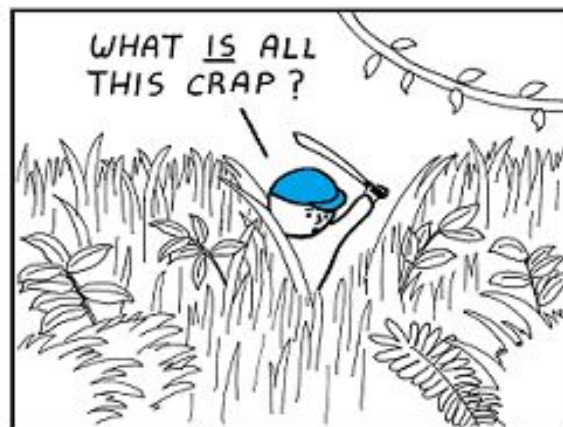
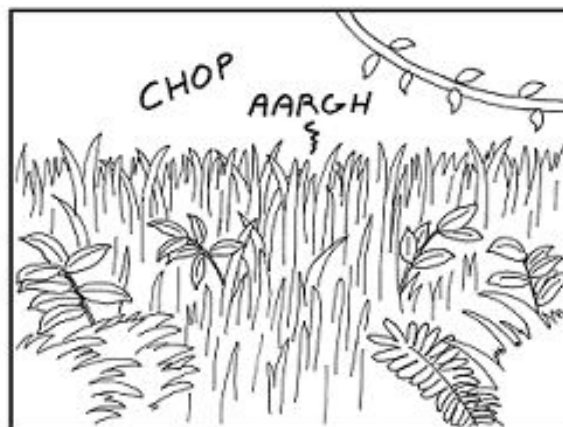
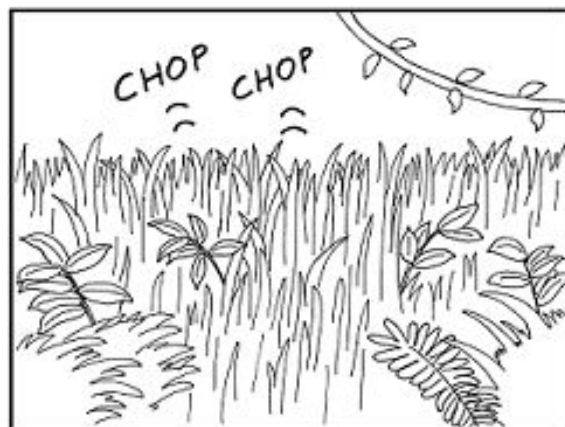
PROFESIONALISMO

## Costo de Poseer código no mantenible



# The ONLY valid measurement of code quality: WTFs/minute





I hate reading  
other people's code.

# Objetivos

**Mantenibilidad**

**Simplicidad**

**Claridad**

**Flexibilidad**

**Legibilidad**

# Nombres Significativos y Pronunciables

```
const d = 3; // Tiempo transcurrido en días
```

## **Preferir Nombres claros a comentarios**

```
const tiempoTranscurridoEnDias = 3;
```

```
const yyyyymmddstr = moment().format('YYYY/MM/DD');
```

## **Y pronunciables**

```
const fechaActual = moment().format('YYYY/MM/DD');
```



# Usar nombres que revelen su intención

```
function obtenerCeldas()  
{  
    let lista1 = new Array();  
    for (let i=0; i<laLista.length; i++)  
        if (laLista[i][0] == 4)  
            lista1.add(laLista[i]);  
    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

```
function obtenerCeldasTransitables()  
{  
    let celdasTransitables = new Array ();  
    tableroJuego.celdas.foreach(function(celda){  
        if (celda.esTransitable())  
            celdasTransitables.add(celda);  
        });  
    return celdasTransitables;  
}
```

# Usar nombres que revelen su intención

```
function getPassableCells()  
{  
  return this.board.cells  
    .filter( cell => cell.isPassable() )  
}
```

# Use Searchable Names

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

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



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

```
// Que significa el 86400000?
```

```
setTimeout(hastaLaInfinidadYMásAlla, 86400000);
```



```
const MILISEGUNDOS_EN_UN_DIA = 8640000;  
setTimeout(hastaLaInfinidadYMásAlla, MILISEGUNDOS_EN_UN_DIA);
```

# 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

conseguirInfoUsuario();

conseguirDataDelCliente();

conseguirRecordDelCliente();

## **Use Solution Domain Names**

# Funciones

- **Funciones/Métodos pequeños**

Deberían tener menos de 8 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

**writeFile(fileName)**

- **Flag**

`show(true)`

Es preferible usar polimorfismo, o crear nuevas funciones

`showFinishedItems()`

`showDraftItems()`



```
function createFile(name, temp) {  
    if (temp) {  
        fs.create(`./temp/${name}`);  
    }  
    else {  
        fs.create(name);  
    }  
}
```



```
function createFile(name) {  
    fs.create(name);  
}
```

```
function createTempFile(name) {  
    createFile(`./temp/${name}`);  
}
```

- **DRY Principle (Don't Repeat Yourself)**

- **The Principle of Least Surprise**

```
Day day = DayDate.StringToDay(String dayName);
```

- **The Boy Scout Rule.**

```
Mostrar(true)
```

# Comentarios

- Es bueno o no es bueno el uso de los comentarios?



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

- **Explica todo**

```
// 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"  
let x;  
// Le sumo 1  
x = x + 1;
```

- **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 fitnessse.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**

# Test Unitarios

- **F.I.R.S.T.**
  - Fast
  - Independent
  - Repeatable
  - Self-Validating
  - Timely

WhatAreWeTesting\_InWhatConditions\_WhatAreExpectedResult

LogIn\_ExistingUsernameWithIncorrectPassword\_ShouldReturnMessageWrongPassword

# Test Unitarios

- **SIEMPRE ESCRIBE CASOS DE PRUEBA AISLADOS**
- **PRUEBA UNA SOLA COSA EN UN SOLO CASO DE PRUEBA**
- **UTILIZA UN ÚNICO MÉTODO DE ASSERT POR CASO DE PRUEBA**
- **UTILIZA UNA CONVENCIÓN DE NOMBRES PARA LOS CASOS DE PRUEBA**
  - isAdult\_AgeLessThan18\_False
  - testIsNotAnAdultIfAgeLessThan18
  - IsNotAnAdultIfAgeLessThan18
  - Should\_ThrowException\_When\_AgeLessThan18
  - When\_AgeLessThan18\_Expect\_isAdultAsFalse
  - Given\_UserIsAuthenticated\_When\_InvalidAccountNumberIsUsedToWithdrawMoney\_Then\_TransactionsWillFail
  - Login\_ExistingUsernameWithIncorrectPassword\_ShouldReturnMessageWrongPassword
- **UTILIZA MENSAJES DESCRIPTIVOS EN LOS MÉTODOS DE ASSERT**
- **MIDE LA COBERTURA DE CÓDIGO PARA ENCONTRAR CASOS DE PRUEBA FALTANTES**

- 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

- IDEs Modernos (IntelliJ, WebStorm, VS Code, Visual Studio, etc)
- Linters (ESLint, SonarLint)
- Formateadores de código (Prettier)
- Herramientas de revisión de código (Gitlab, Github)
- Herramientas de análisis estático (SonarQube, CodeClimate, y Coverity Scan)
- Herramientas de pruebas unitarias y cobertura de código (Jest, Mocha, JUnit)
- Herramientas de integración continua (Jenkins, Gitlab CI, Azure Devops)

# Bibliografía

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