Diseño de Código

FI.UBA

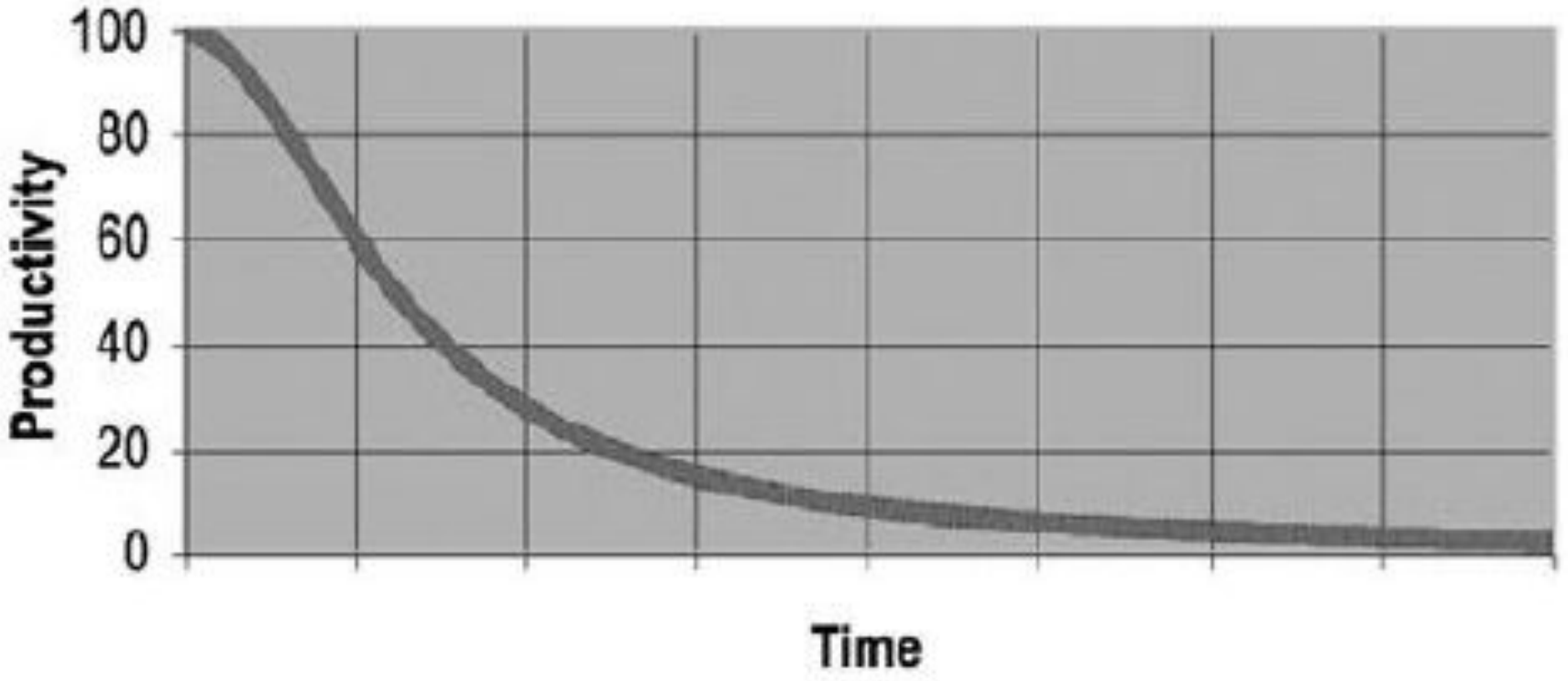
Ingeniería de Software

.UBAfiuba
FACULTAD DE INGENIERÍA

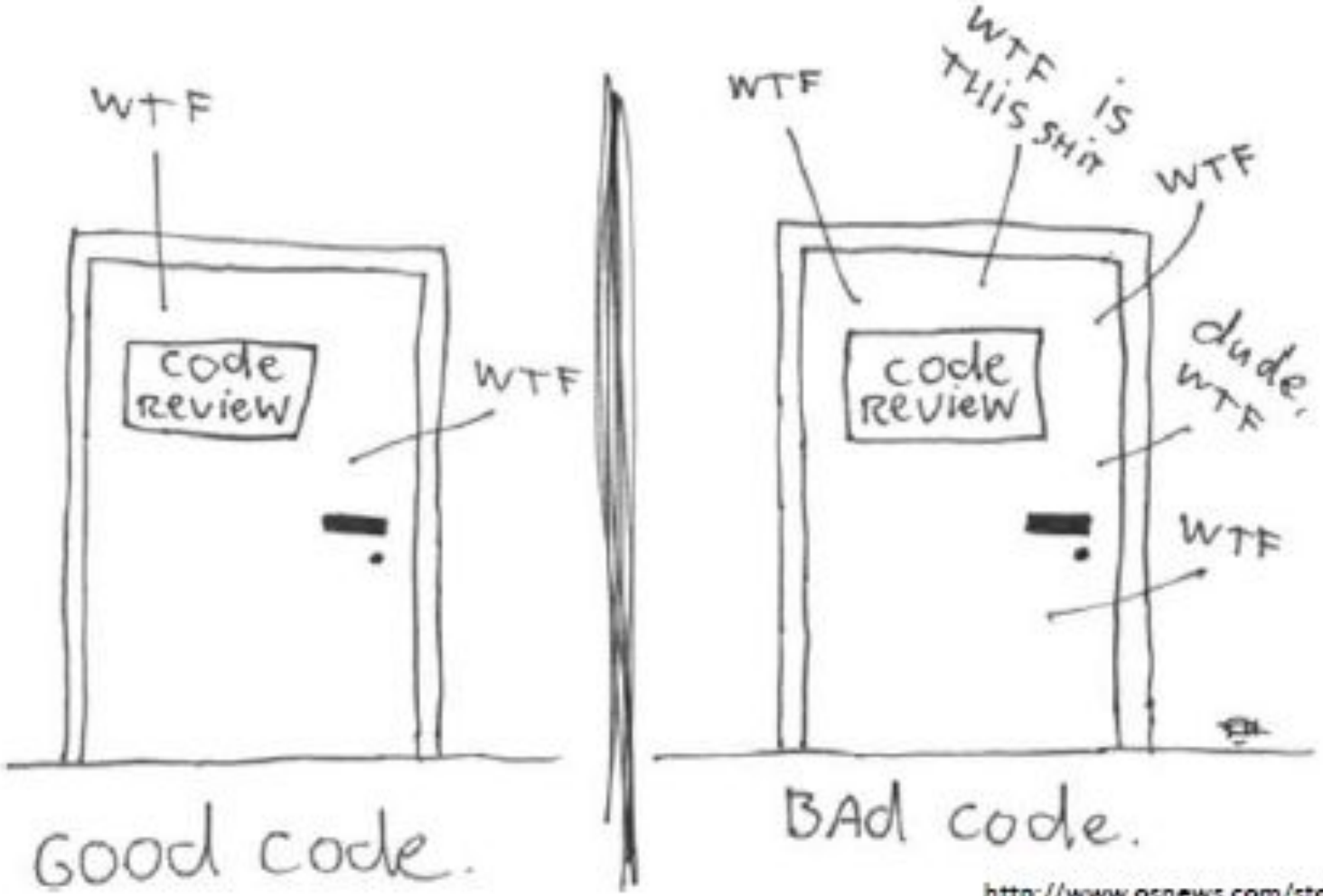
ORGULLO

PROFESIONALISMO

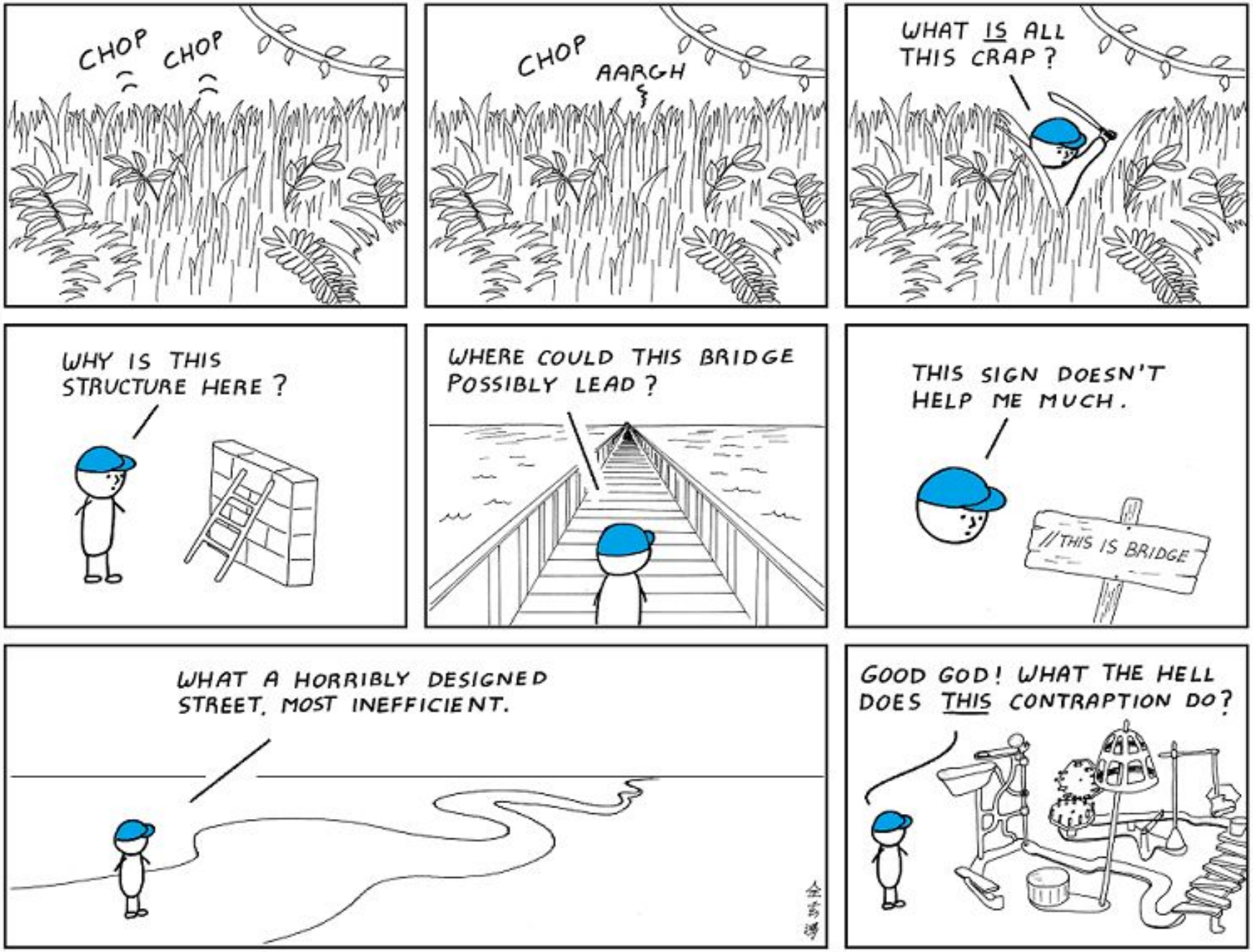
Costo de Poseer código no mantenible



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 y Pronunciables

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

Preferir Nombres claros a comentarios

```
const tiempoTranscurridoEnDias = 3;
```

```
const yyyymmdstr = 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;
}
```

```
letrealDaysPerIdealDay = 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(hastaLaInfinidadYMasAlla, 86400000);
```

```
const MILISEGUNDOS_EN_UN_DIA = 8640000;
setTimeout(hastaLaInfinidadYMasAlla, 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;
   }
}
```

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

Class Names, Method Names

Pick One Word per Concept

fetch, retrieve, get

conseguirInfoUsuario();
conseguirDataDelCliente();
conseguirRecordDelCliente();

Use Solution Domain Names

- **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)
```

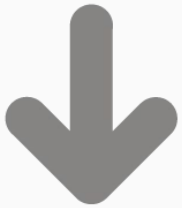
- **KISS**

Comentarios

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

- **Explíca 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 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**

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 cómo 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 y Editores de Código

- **IntelliJ IDEA** – Potente IDE para Java y otros lenguajes.
- **WebStorm** – IDE optimizado para JavaScript y TypeScript.
- **VS Code** – Editor ligero y altamente extensible.
- **Visual Studio** – IDE completo para .NET y más.
- **Fleet** – IDE moderno y ligero de JetBrains.
- **Zed** – Editor rápido y colaborativo.
- **Cursor** – Fork de VS Code con IA integrada para mejorar código.

IA para Autocompletado y Generación de Código

- **GitHub Copilot** – Basado en OpenAI, ayuda a generar código.
- **Codeium** – Alternativa gratuita a Copilot con IA avanzada.
- **Tabnine** – IA que sugiere código basado en patrones previos.
- **Cursor** – Integra IA para mejorar código y refactorizar.

Linters y Formateadores

- **ESLint** – Linter para JavaScript y TypeScript.
- **SonarLint** – Detecta errores y malas prácticas en tiempo real.
- **Pylint** – Análisis de código Python.
- **Checkstyle** – Revisión de estilo en Java.
- **PMD** – Análisis estático para Java y otros lenguajes.
- **Prettier** – Formateador de código automático.
- **Biome** – Reemplazo moderno para ESLint y Prettier, más rápido.
- **Rome** – Alternativa todo-en-uno para linting y formateo.

Herramientas útiles

Herramientas de Revisión de Código

- **GitHub** – Plataforma de control de versiones y revisiones de código.
- **GitLab** – Alternativa a GitHub con CI/CD integrado.
- **Reviewpad** – Automatiza revisiones de código en GitHub y GitLab.
- **Graphite** – Mejora la gestión de Pull Requests en GitHub.

Análisis Estático de Código y Calidad

- **SonarQube** – Análisis estático para detectar vulnerabilidades y problemas.
- **CodeClimate** – Evalúa calidad de código y deuda técnica.
- **Coverity Scan** – Detecta errores de seguridad en código.
- **DeepSource** – Alternativa moderna a SonarQube con mejor UX.
- **CodeScene** – Agrega análisis de deuda técnica y patrones de trabajo en equipos.

Herramientas de Pruebas Unitarias y Cobertura de Código

- **Jest** – Testing para JavaScript.
- **Mocha** – Framework de pruebas para Node.js.
- **JUnit** – Framework de pruebas unitarias para Java.
- **Mutation Testing (StrykerJS, MutPy, PITest)** – Evalúa la calidad de los tests modificando el código.
- **Playwright** – Alternativa moderna para testing end-to-end.

Integración Continua y Automatización

- **Jenkins** – Herramienta de CI/CD.
- **GitLab CI** – CI/CD integrado en GitLab.
- **Azure DevOps** – Plataforma de CI/CD de Microsoft.
- **Dagger** – Alternativa moderna a Jenkins/GitLab CI para flujos CI/CD portables.
- **Earthly** – Construcción de CI/CD moderna con scripting más limpio.

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)