2024

Clean Code

Disminuyendo la carga cognitiva del código

¿Quién soy?

- Leonardo Micheloni
- Programador +20 años
- Microsoft MVP 9 años
- CSM desde 2012
- Arquitecto y tech lead en Tokiota Madrid

@leomicheloni

Sponsors

NTT DATA

plain concepts

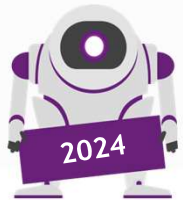
intelequia

prodware

TOKIOTA

LIDL SCRM INTERNATIONAL HUB

ilitia

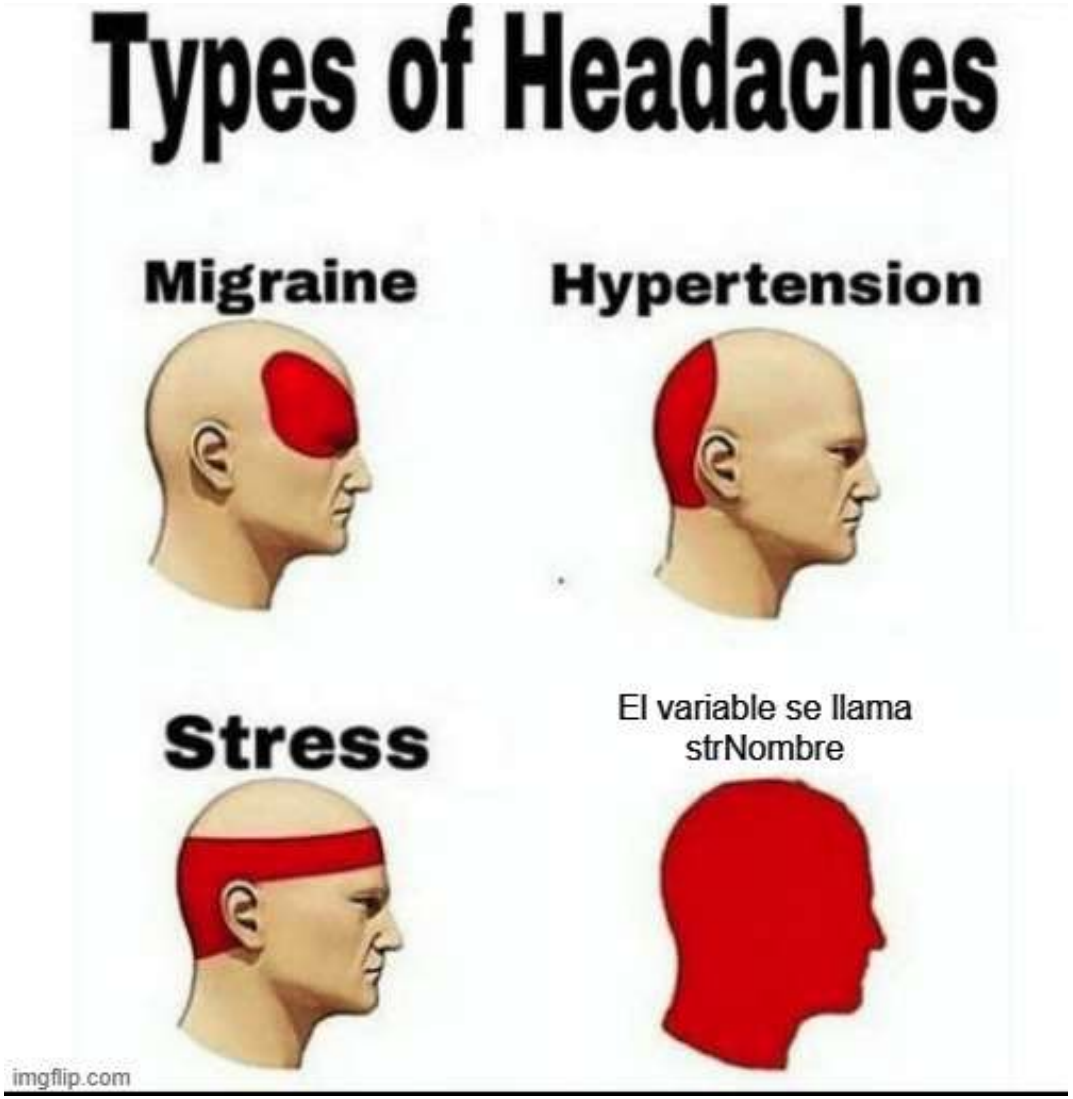


#netcoreconf

¿Disminuir la carga cognitiva del código?

- Detenernos a pesar qué hace un if
- Nos cuesta encontrar el código que buscamos
- No notar claramente la intención de una función

- Nos vamos cansando de analizar
- Nos perdemos con facilidad
- Odiamos todo


Types of Headaches
Migraine
Hypertension
Stress
El variable se llama
strNombre
imgflip.com

Agenda

01 DRY - KISS - YAGNI

02 NAMING

03 CONDICIONALES

04 FUNCIONES

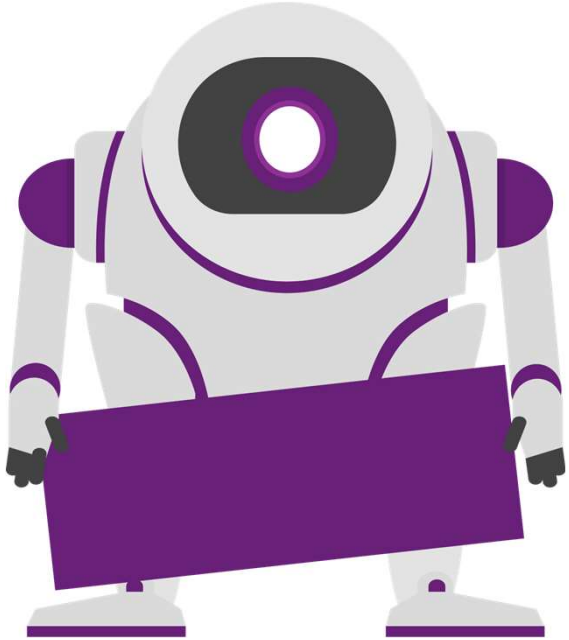
05 COMENTARIOS

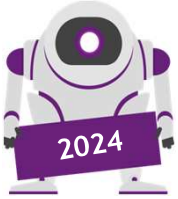
06 ALGUNA COSA MÁS


Contenido de la charla
CÓDIGO SIN CONTEXTO
REFLEXIONES
AI
HERRAMIENTAS DE MODA
MEMES

Title Example

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse et est tempus, semper mi vitae, rutrum enim. Aliquam ac egestas eros. Ut volutpat elit ligula. Integer molestie odio diam, a sollicitudin arcu fermentum non. Donec euismod, ligula sed efficitur ornare, lacus erat dapibus sapien, ac gravida lectus ante ac nisi. Suspendisse laoreet pretium ornare. Donec fringilla non odio nec molestie. Nulla hendrerit consequat tellus fringilla laoreet. Phasellus blandit bibendum luctus. Etiam consequat diam eu porta porta. Cras sit amet porta est. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. In non ullamcorper nisi.

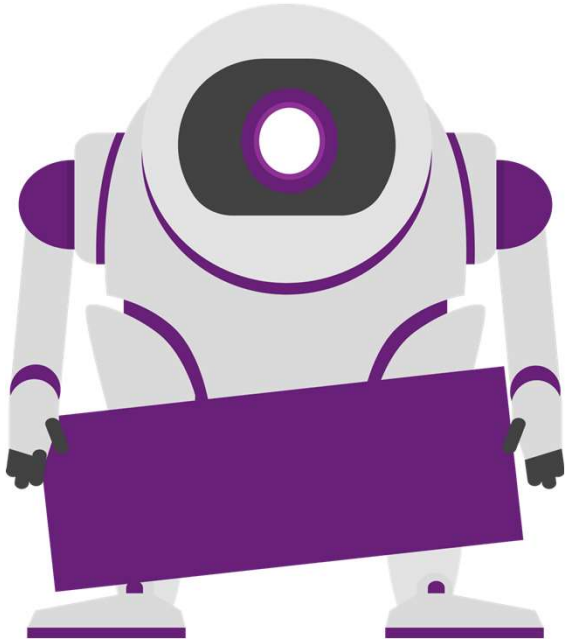


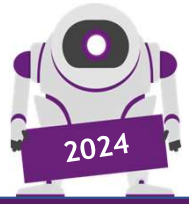


#netcoreconf

Title example

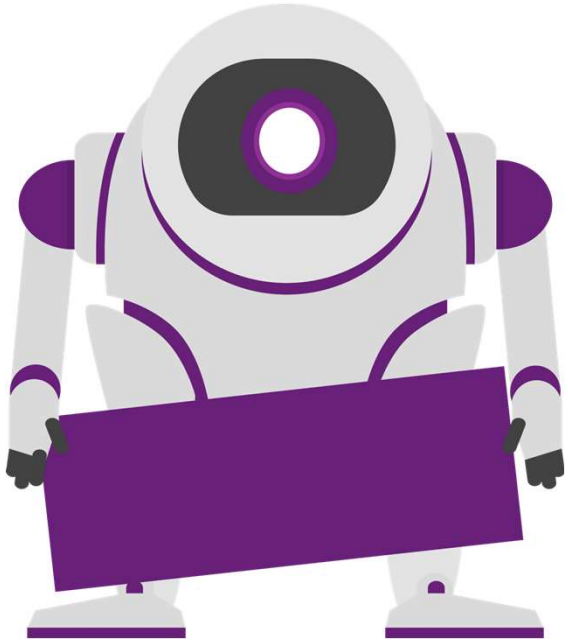
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse et est tempus, semper mi vitae, rutrum enim. Aliquam ac egestas eros. Ut volutpat elit ligula. Integer molestie odio diam, a sollicitudin arcu fermentum non. Donec euismod, ligula sed efficitur ornare, lacus erat dapibus sapien, ac gravida lectus ante ac nisi. Suspendisse laoreet pretium ornare. Donec fringilla non odio nec molestie. Nulla hendrerit consequat tellus fringilla laoreet. Phasellus blandit bibendum luctus. Etiam consequat diam eu porta porta. Cras sit amet porta est. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. In non ullamcorper nisi.

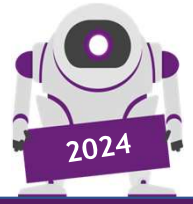




¿Por qué son importantes estos conceptos?

- Un framework / lenguaje / library no es más que una herramienta
- Se habla de ser bueno usando x pero no de ser buen programador
- Es cada vez más común mezclar tecnologías y herramientas
- Las aplicaciones evolucionan cada vez más rápido
- No siempre somos nosotros quienes comenzamos el código

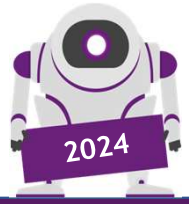


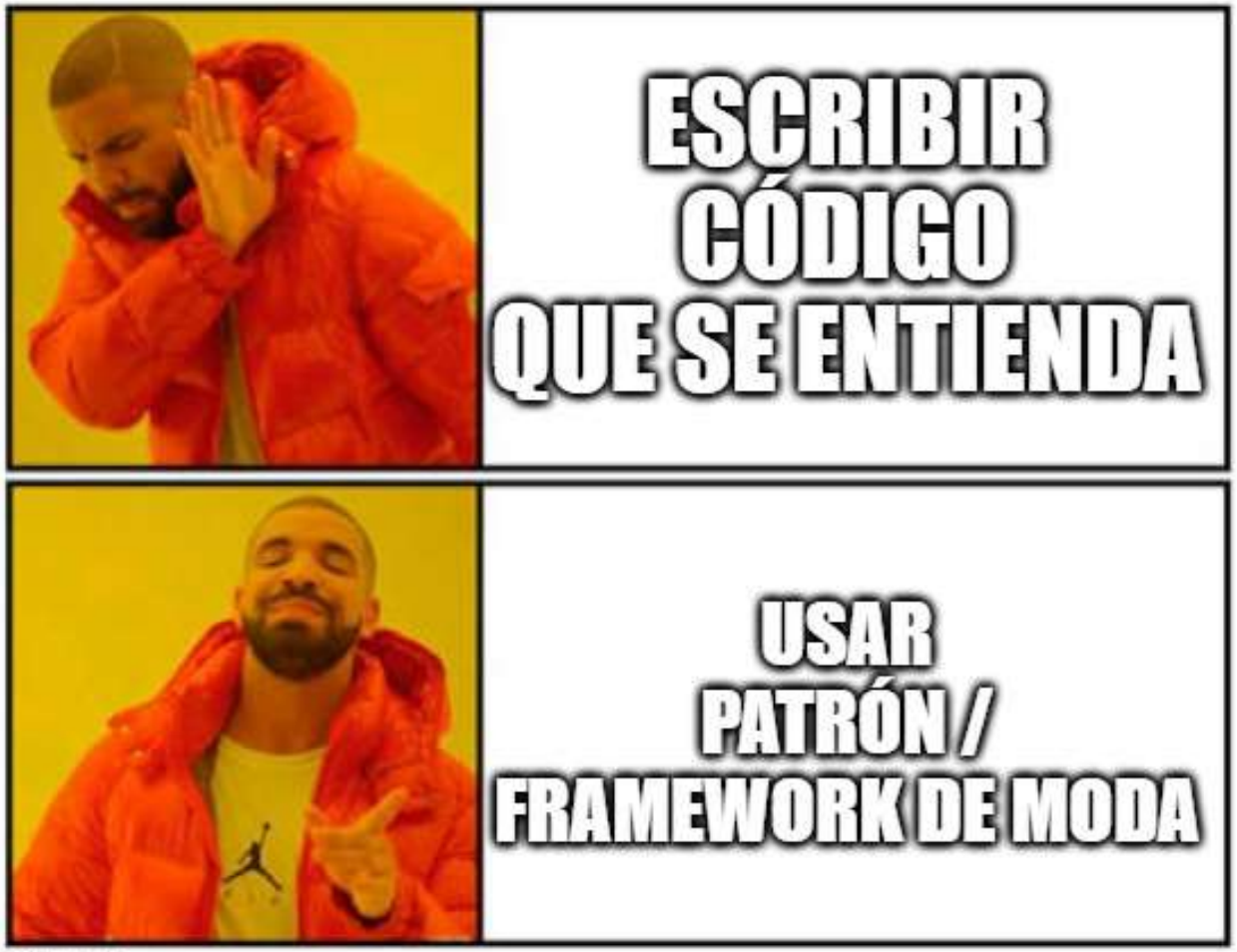


"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."

Martin Fowler

No es Martin Fowler




ESCRIBIR
CÓDIGO
QUE SE ENTIENDA
USAR
PATRÓN /
FRAMEWORK DE MODA
imgflip.com

DRY

- **Don’t repeat yourself principle**
- Es similar a la normalización de bases de datos
- Copy + Paste oriented programming
- Hay más líneas de código para mantener

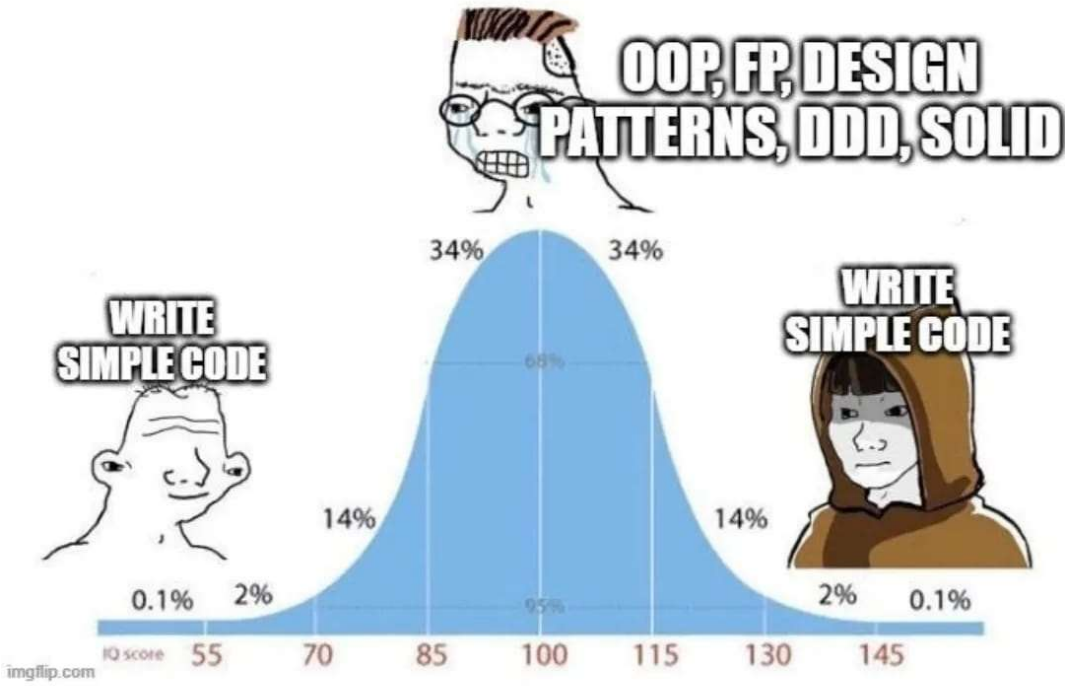
- Peeero..

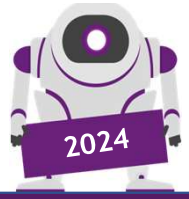
KISS

- **Keep it simple stupid**
- Hay un momento en nuestra evolución que nos gusta ser complicados
- La simplicidad es belleza

YAGNI

- **You ain’t gonna need it**
- Voy a hacer tal cosa por las dudas
- No se extremistas






```cs
using System;
using System.Threading;
using System.Threading.Tasks;
using MediatR;

namespace SaludoGlobalizado
{
    // Definimos una interfaz para la abstracción del lo
    4 references
    public interface ILogService
    {
        3 references
        Task LogAsync(string mensaje);
    }

    // Implementación concreta del servicio de log
    1 reference
    public class ConsoleLogService : ILogService
    {
        3 references
        public async Task LogAsync(string mensaje)
        {
            await Task.Run(() => Console.WriteLine($"[LOG]: {mensaje}"));
        }
    }

    // Definimos un comando para solicitar el saludo en un idioma específico
    3 references
    public class ObtenerSaludoCommand : IRequest<string>
    {
        2 references
        public string Idioma { get; set; }
    }

    // Manejador para el comando de obtener saludo
    3 references
    public class ObtenerSaludoCommandHandler : IRequestHandler<ObtenerSaludoCommand, string>
    {
        3 references
        private readonly ILogService _logService;

        1 reference
        public ObtenerSaludoCommandHandler(ILogService logService)
        {
            _logService = logService;
        }

        0 references
        public async Task<string> Handle(ObtenerSaludoCommand request, CancellationToken cancellationToken)
        {
            var idiomaEncontrado = ObtenerIdioma(request.Idioma);
            if (idiomaEncontrado != null)
            {
                await _logService.LogAsync($"Saludo en {idiomaEncontrado.Nombre}: {idiomaEncontrado.Saludo}");
                return idiomaEncontrado.Saludo;
            }
            else
            {
                await _logService.LogAsync("No se encontró el idioma especificado.");
                return "No se encontró el idioma especificado.";
            }
        }

        private Idioma ObtenerIdioma(string idioma)
        {
            // Implementación ficticia para obtener el idioma (similar a la versión anterior)
            return idioma.Equals("Inglés", StringComparison.OrdinalIgnoreCase)
                ? new Idioma { Nombre = "Inglés", Saludo = "Hello, World!" }
                : idioma.Equals("Español", StringComparison.OrdinalIgnoreCase)
                    ? new Idioma { Nombre = "Español", Saludo = "¡Hola, Mundo!" }
                    : null;
        }
    }

    // Configuración y ejecución
    0 references
    class Program
    {
        0 references
        static async Task Main(string[] args)
        {
            // Creamos una instancia del servicio de log
            var logService = new ConsoleLogService();

            // Configuramos MediatR con un service locator
            var mediator = ConfigureMediator(logService);

            // Solicitamos el saludo en un idioma específico (por ejemplo, inglés)
            var idiomaElegido = "Inglés";
            var saludo = await mediator.Send(new ObtenerSaludoCommand { Idioma = idiomaElegido });

            // No necesitamos Console.WriteLine aquí, ya que el log se maneja internamente
        }

        // Configuración de MediatR con un service locator
        1 reference
        static IMediator ConfigureMediator(ILogService logService)
        {
            var serviceFactory = new ServiceFactory(type =>
            {
                if (type == typeof(ObtenerSaludoCommandHandler))
                    return new ObtenerSaludoCommandHandler(logService);

                // Agrega más manejadores aquí si es necesario

                throw new InvalidOperationException($"Manejador no registrado para {type}");
            });

            return new Mediator(serviceFactory);
        }
    }
}
```

Simple, direct, prose

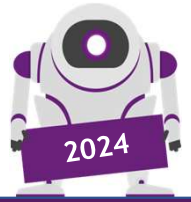
Clean code is simple
and direct

Clean code reads like
well-written prose

Grady Booch

Code Smells

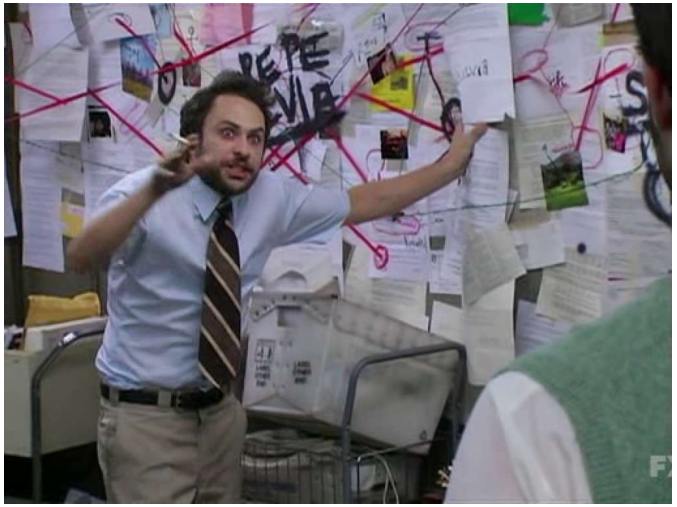
- Olores de código
- Indican que el código tiene potenciales problemas
  - Para ser mantenido
  - Para escalar
  - Para evolucionar
- Es propenso a errores



Naming

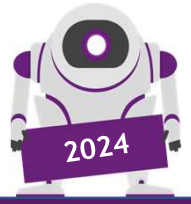
Clean code: Naming

- Deben ser claros en su intención
- Deben indicar responsabilidad única
- No deben tener prefijos o subfijos
- Las variables booleanas deben responder preguntas positivas



Clean code: Naming

- Smells
  - AND OR IF
  - El código tiene “side effects”
  - Tipos de datos en el nombre
  - Booleanos que no responden una pregunta
  - Variables o métodos que no son simétricos
  - Funciones con efectos secundarios no descriptos por su nombre
  - Las propiedades de un objeto comienzan por el nombre del objeto



Naming intention

```
//// Naming intention
// Mal
string d = DateTime.Now.ToString("yyyy/MM/dd");
// bien
string currentDate = DateTime.Now.ToString("yyyy/MM/dd");

// Mal
void Save()
{
    // ...
}
// Bien
void SaveUserStatus()
{
    // ...
}

// Mal
string[] a = { "Ford", "Rena", "Citroen", "Mazda" };
foreach (var element in a)
{
    // ...
}
// Bien
string[] carMakers = { "Ford", "Rena", "Citroen", "Mazda" };
foreach (var maker in carMakers)
{
    // ...
}
```

Symmetric naming

```
//// Naming, be symmetric
// Mal
public void Start()
{
    // ...
}
public void Finish()
{
    // ...
}

// Bien
public void Start()
{
    // ...
}
public void Stop()
{

}

// Mal
bool on = true;
bool down = false;

// Bien
bool on = true;
bool off = false;
```

Avoid redundant naming

```
//// Redundant naming
// Mal
1 reference
public class Car
{
    0 references
    public string carMake = "Honda";
    0 references
    public string carModel = "Accord";
    0 references
    public string carColor = "Blue";
}

// Bien
1 reference
public class Car
{
    0 references
    public string make = "Honda";
    0 references
    public string model = "Accord";
    0 references
    public string color = "Blue";
}
```

Booleans should answer questions

```
// Naming, boolean should answer question
// Mal
public void Valid()
{
    // ...
}
// Bien
public void IsValidSubmission()
{
    // ...
}

// Mal
bool valid = true;
// Bien
bool isValid = true;

// Mal
bool registered = false;
// Bien
bool isRegistered = false;

// Mal
bool notValid = false;
// Bien
bool isValid = true;



// don't double negate
// Mal
if (!notValid)
{
    // ...
}

// Bien
if (isValid)
{
    // ...
}
```

Naming variables

- Don’t use abbreviations
- Don’t use prefixes of subfixes
- Use noun + verb in function

```
//// Naming, don't use abbreviations
// Mal
var usrPwd = // ....
// Bien
var userPassword = //
```

```
//// Naming, don't use prefixes or suffixes
string strName = "Chespirito";

// Mal

if (bExiste == true)
{
    // ...
}
```

Naming functions

- Don’t use AND, OR, etc.
- Use noun + verb in function

```
// Naming, don't use AND, OR, IF
public void ValidateAndSave(User user)
{
    // ...
}

public void AddOrCreate(File file)
{
    // ...
}

public void SaveOrUpdate()
{
    // ...
}
```

```
//// Naming use noun + verb in functions
// Bad
void Do(User u, Credentials c){
    // ...
}
// Good
void RegisterUser(User user, Credentials credentials){
    // ...
}

// Bad
void get(){
    // ....
}
// Good
void GetRegisteredUsers(){
    // ....
}
```

Clean Code: Naming

- Soluciones
  - Verbalizar con un amigo
  - O con un pato de goma
  - Refactorizar hasta que el nombre sea “limpio”

Condicionales

Clean Code: Condicionales

- Comparar explícitamente
- Usar condicionales positivos (evitar la doble negación)
- Asignar implícitamente
- Evitar "magic numbers"
- Evitar condicionales complejos

Clean Code: Condicionales

- Smells
  - Ser “antinegativo”
  - Magic numbers
  - Asignaciones de booleanos en condiciones
  - Comentarios sobre una condición
  - Condiciones difícil de comprender

Avoid magic numbers

```
//// Conditionals, don't use magic numbers
// Mal
if(user.age > 18){
    // ...
}
// Bien
int validAge = 18;
if(user.age > validAge){
    // ...
}

// Mal
Thread.Sleep(86400000);
// Bien
const int MILLISECONDS_IN_A_DAY = 86400000;
Thread.Sleep(MILLISECONDS_IN_A_DAY);

// Mal
if(userStatus == -1){ // new user
    // ....
}
// Bien
int UserStatusNew = -1;
if(userStatus == UserStatusNew){
    // ....
}
```

Conditionales

- Must be positive
- Don’t be “antinegative”


```
//// Conditionals, be "positive"
// Mal
if(isNotValid())
{
    // ...
}
// Bien
if(isValid())
{
    // ...
}
```

```
//// Conditionals, don't be "anti-negative"
// Mal
if(!IsNotValid())
{
    // ...
}
// Bien
if(IsValid())
{
    // ...
}
```

Conditionales

- Use direct comparison
- Use ternary operator

```
//// Conditionals, use direct comparison
// Mal
if(isValid() == true)
{
    // ...
}
// Bien
if(isValid())
{
    // ...
}
```

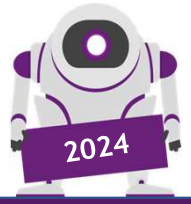
```
//// Conditionals, assign using ternary operator
string message = "";

if (IsValid())
{
    validationOk = "Welcome!";
}
else
{
    validationOk = "Go Away";
}

// Bien
string validationOk = IsValid() ? "Welcome!" : "Go Away";
```

Use functions to clarify

```
//// Conditional, use functions to clarify conditions
// Mal
if (fsm.state == "fetching" && isEmpty(listNode))
{
    // ...
}
// Bien
public bool ShouldShowSpinner(Fsm fsm, ListNode listNode)
{
    return fsm.state == "fetching" && IsEmpty(listNode);
}
if (ShouldShowSpinner(fsmInstance, listNodeInstance))
{
    // ...
}
```

Clean Code: Condicionales

- Soluciones
  - Siempre usar booleanos que respondan preguntas positivas
  - Poner magic numbers en variables
  - Crear funciones si la condición es una regla de negocio
  - Utilizar variables intermedias

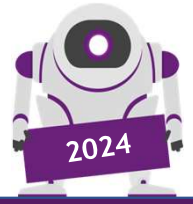



ONe
WEEK
LAteR...

Funciones

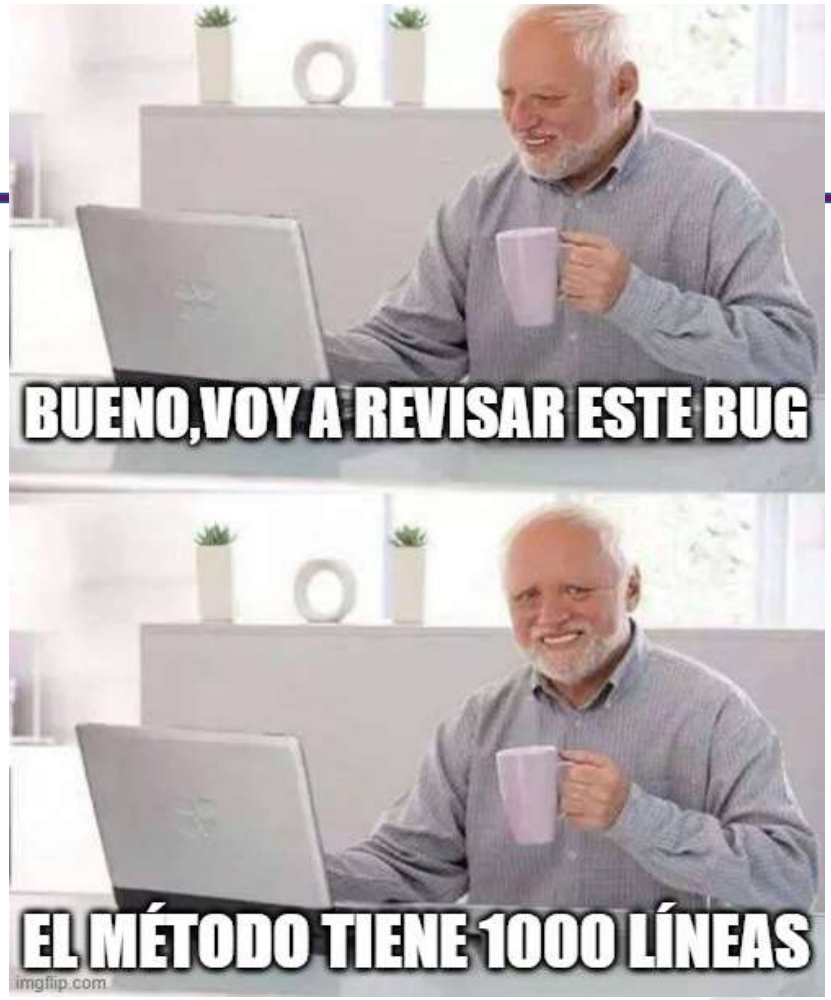
Clean Code: Funciones

- Primitive obsession
- Evitar Arrow Code
- Evitar funciones con demasiada responsabilidad
- Evitar funciones con efectos secundarios
- Evitar muchos argumentos
- Evitar funciones con flag arguments



Clean code: Funciones

- Smells
  - Hay que hacer scroll para leerlas, vertical / horizontal
  - Tienen demasiados parámetros
  - Intención poco clara
  - Cambian con mucha frecuencia



Primitive obsession

```
//// Classes, avoid primitive return types, use classes
// Bad
public object RegisterUser(User user)
{
    // ...
    return json;
}
// Good
public UserRegistrationResult RegisterUser(User user)
{
    return new UserRegistrationResult
    {
        Success = true,
        UserId = 123,
        WelcomeDiscount = 10
    };
}

2 references
public class User
{
    // properties and methods of User class
}

2 references
public class UserRegistrationResult
{
    1 reference
    public bool Success { get; set; }
    1 reference
    public int UserId { get; set; }
    1 reference
    public int WelcomeDiscount { get; set; }
}
```

Arrow code

```
//// Conditionals, avoid arrow code
// MAL
string result = "";

if (!user.IsRegistered())
{
    if (user.Name.Length < 3)
    {
        result = "the name is too short";
    }
    else
    {
        if (user.Password.Length < 4)
        {
            result = "the password is too short";
        }
        else
        {
            if (user.Age < 18)
            {
                result = "you must be over 18 in order to register my friend";
            }
            else
            {
                if (user.Gender == "")
                {
                    result = "A gender must be specified";
                }
                else
                {
                    if (user.Country == "")
                    {
                        result = "Please select a country from the list";
                    }
                    else
                    {
                        if (user.HasValidEmail())
                        {
                            result = "Welcome man";
                        }
                        else
                        {
                            result = "The email is not valid";
                        }
                    }
                }
            }
        }
    }
}
else
{
    result = "This user is already registered";
}
```

```
// Bien - guard clauses
if (user.IsRegistered())
    return "This user is already registered";

if (user.Name.Length < 3)
    return "The name is too short";

if (user.Password.Length < 4)
    return "the password is too short";

if (user.Age < 18)
    return "you must be over 18 in order to register my friend";

if (user.Gender == "")
    return "A gender must be specified";

if (user.Country == "")
    return "Please select a country from the list";

if (!user.HasValidEmail())
    return "The email is not valid";

return "Welcome man";
```

Too many arguments

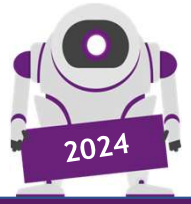
```
//// Functions, avoid too many arguments, use clases instead
// Mal
public void Register(string name, string lastName, int age, string email, string country)
{
    // ...
}

// Bien
2 references
public class User
{
    1 reference
    public string Name { get; set; }
    1 reference
    public string LastName { get; set; }
    1 reference
    public int Age { get; set; }
    1 reference
    public string Email { get; set; }
    1 reference
    public string Country { get; set; }

    0 references
    public User(string name, string lastName, int age, string email, string country)
    {
        Name = name;
        LastName = lastName;
        Age = age;
        Email = email;
        Country = country;
    }
}

public void RegisterUser(User user)
{
    // ...
}
```

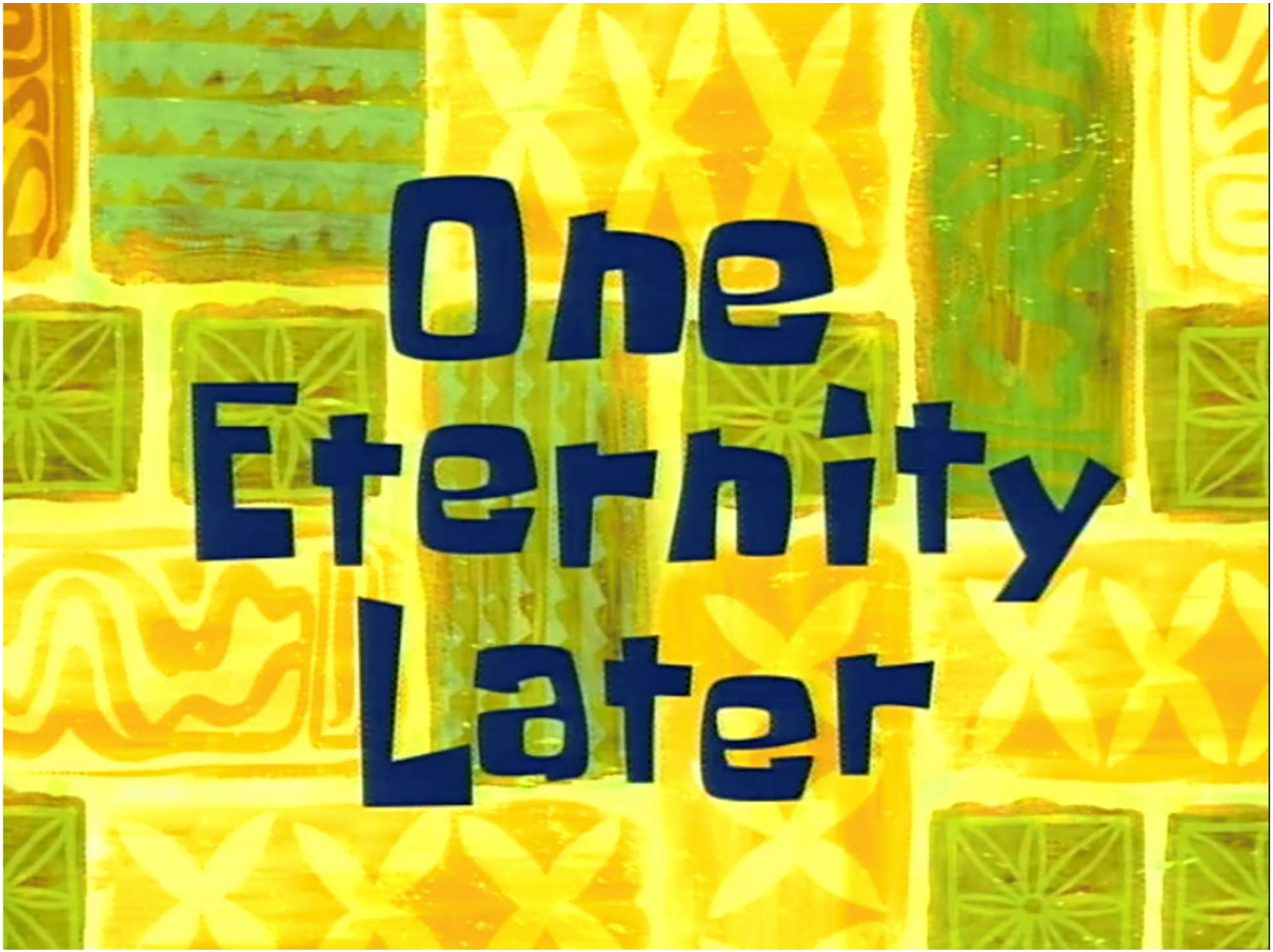
Flag arguments

```
//// Functions, don't use flag arguments
  // Bad
  void CreateFile(string name, bool temp)
  {
    if (temp)
    {
      fs.Create($"./temp/{name}");
    }
    else
    {
      fs.Create(name);
    }
  }
  // Good
  void CreateFile(string name)
  {
    fs.Create(name);
  }
  void CreateTempFile(string name)
  {
    CreateFile($"./temp/{name}");
  }
```

Clean Code: Funciones

- Soluciones
  - Guard clauses
  - Fail fast
  - Extract method
  - Nuevas clases
  - Rename

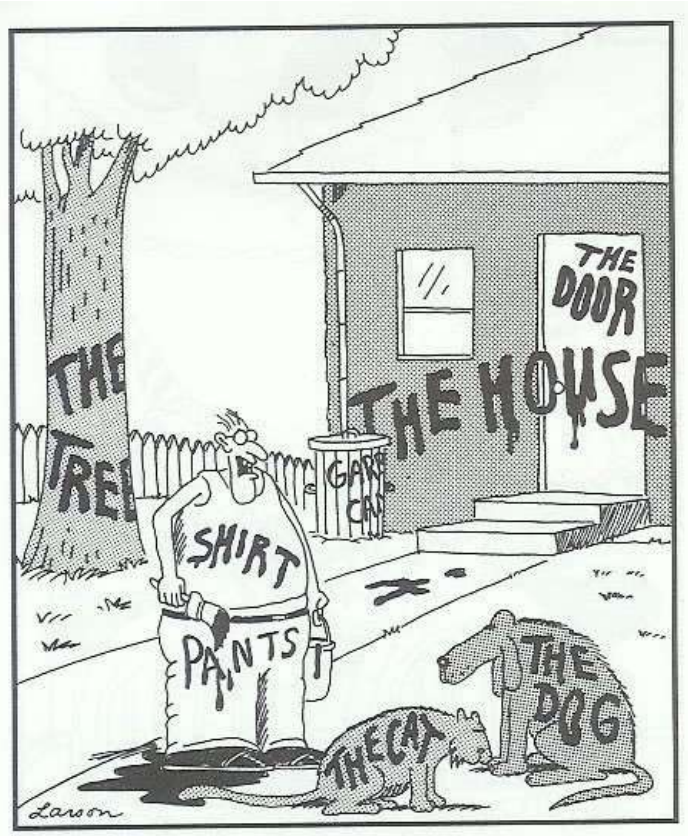



One
Eternity
Later

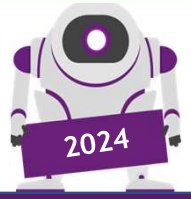
Comentarios

Clean Code: Comentarios

- Pueden indicar que nuestro código no es claro
- Evitar comentarios obvios
- Evitar comentarios desactualizados

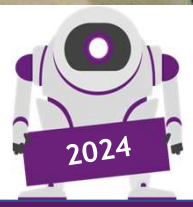


"Now! … *That* should clear up a few things around here!"



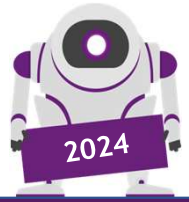
Clean Code: Comentarios

- Smells
  - Los comentarios son más grandes que el código
  - El comentario dice lo mismo que el nombre de la función
  - El comentario intenta explicar algo que el código debería
  - El comentario está desactualizado
  - El comentario indica miedo



Clean Code: soluciones

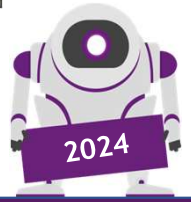
- Al escribir un comentario pensar por qué lo hacemos
- Cambiar la función / variable para que indique su intención
- Borrar comentarios en condiciones y extraer la condición



Keep improving

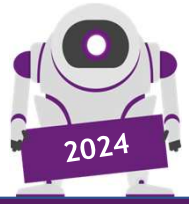
Refactor

- Cambiar el código sin modificar su comportamiento
- Hacerlo más legible
- Lo detectamos gracias a Code Smells, Patrones, DRY, etc.
- Aplicar algunos de los conceptos
  - KISS
  - DRY
  - SOLID
- Baby steps
- El mejor refactor es el que quita código



Mind set

- Mirar el código
- No odiar
- Respect
- Baby steps



Mantenimiento

- Refactoring
- Baby steps
- No aceptar ventanas rotas
- Code reviews
- Pair programming

"Always leave the code you're editing a little better than you found it."
Robert C. Martin

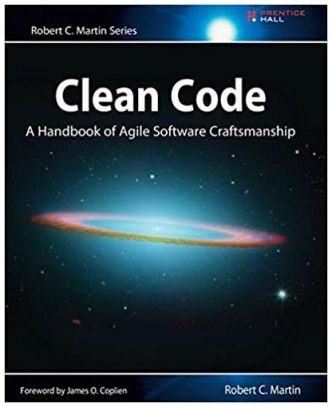
John Lennon (tal vez)

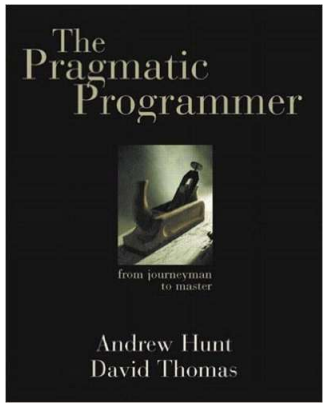
“Life is what happens to you while you’re busy trying to understand your own code”

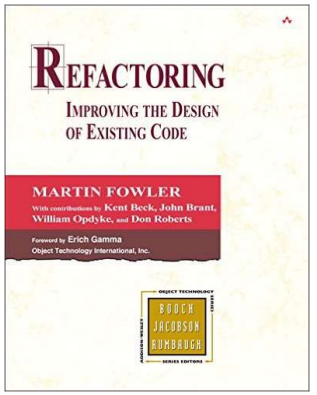
¿Preguntas?

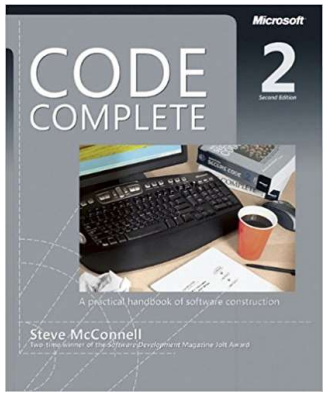
Referencias

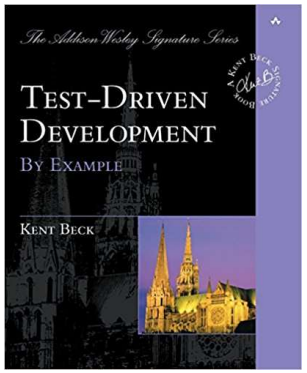
- https://github.com/leomicheloni/sabados-tech-clean-code
- https://app.pluralsight.com/library/courses/writing-clean-code-humans
- https://www.amazon.es/Refactoring-Improving-Design-Existing-Technology/dp/0201485672/
- https://www.amazon.es/Clean-Code-Handbook-Software-Craftsmanship/dp/0132350882
- https://www.amazon.es/Code-Complete-Practical-Costruction-Professional/dp/0735619670/
- https://www.amazon.es/Pragmatic-Programmer-Journeyman-Master/dp/020161622X/
- https://www.amazon.es/Head-First-Design-Patterns-Freeman/dp/0596007124/
- https://github.com/tcorral/Refactoring_Patterns











Sponsors

NTT DATA

plain concepts

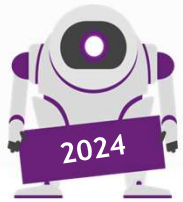
intelequia

prodware

TOKIOTA

LIDL SCRM INTERNATIONAL HUB

ilitia



#netcoreconf

More information:
info@netcoreconf.com
@Netcoreconf

Visit on:
netcoreconf.com