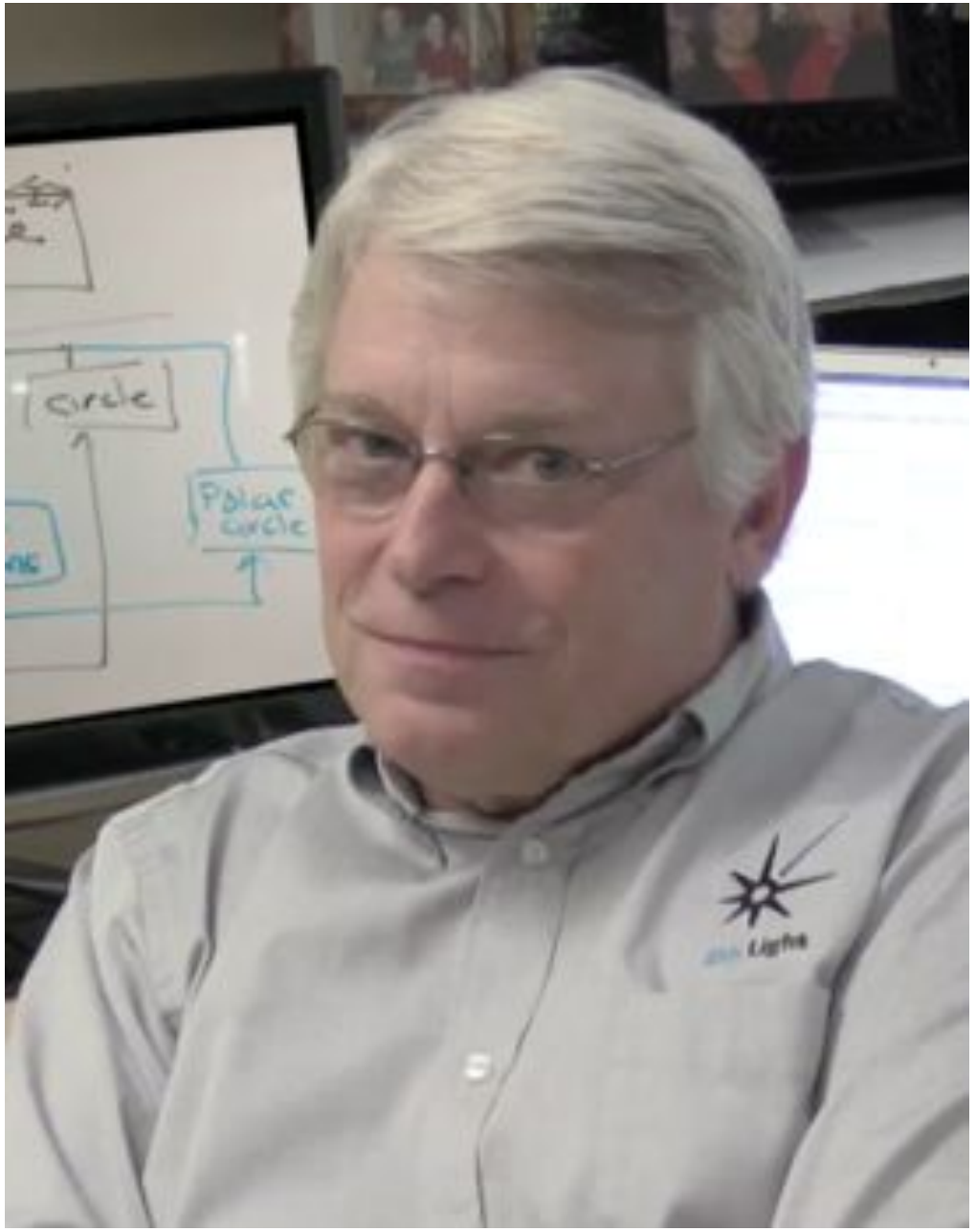

SOLID

S	O	L	I	D
SRP	OCP	LSP	ISP	DIP
Single Responsability Principle	Open / Closed Principle	Liskov Substitution Principle	Interface Segregation Principle	Dependency Inversion Principle

Born	Robert Cecil Martin 5 December 1952 (age 68)
Nationality	American
Other names	"Uncle Bob" Martin
Occupation	Software engineer, instructor
Known for	Agile Manifesto, SOLID principles
Children	4
Website	cleancoder.com



Single-responsibility principle

```
class Report
{
    public string Text { get; set; }

    public void GoToFirstPage()
    {
        Console.WriteLine("Go to the first page");
    }

    public void GoToLastPage()
    {
        Console.WriteLine("Go to the last page");
    }

    public void GoToPage(int pageNumber)
    {
        Console.WriteLine("Go to page {0}", pageNumber);
    }

    public void Print()
    {
        Console.WriteLine("Print report");
        Console.WriteLine(Text);
    }
}
```

```
class Report
{
    public string Text { get; set; }

    public void GoToFirstPage()...

    public void GoToLastPage()...

    public void GoToPage(int pageNumber)...

    public void Print()...

    public void PrintToPDF()...

    public void PrintToPrinter()...
}
```

```
class Report
{
    public string Text { get; set; }

    public void GoToFirstPage()...

    public void GoToLastPage()...

    public void GoToPage(int pageNumber)...

    public void Print()...

    public void PrintToPDF()...

    public void PrintToPrinter()...

}
```

```
class Report
{
    public string Text { get; set; }

    public void GoToFirstPage()...

    public void GoToLastPage()...

    public void GoToPage(int pageNumber)...
}

class Printer
{
    public void Print(Report report)...

    public void PrintToPDF(Report report)...

    public void PrintToPrinter(Report report)...
}
```

```
class Report
{
    public string Text { get; set; }

    public void GoToFirstPage()...

    public void GoToLastPage()...

    public void GoToPage(int pageNumber)...
}

class Printer
{
    public void Print(Report report)...

    public void PrintToPDF(Report report)...

    public void PrintToPrinter(Report report)...
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Report report = new Report();
        report.Print();
    }
}
```

```
interface IPrinter
{
    void Print(string text);
}

class ConsolePrinter : IPrinter
{
    public void Print(string text)
    {
        Console.WriteLine(text);
    }
}
```

```
interface IPrinter
{
    void Print(string text);
}
```

```
class ConsolePrinter : IPrinter
{
    public void Print(string text)...
}
```

```
class PDFPrinter : IPrinter
{
    public void Print(string text)...
}
```

```
class Report
{
    public string Text { get; set; }

    public void GoToFirstPage()...

    public void GoToLastPage()...

    public void GoToPage(int pageNumber)...

    public void Print(IPrinter printer)...
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Report report = new Report();
        IPrinter consolePrinter = new ConsolePrinter();
        IPrinter pdfPrinter = new PDFPrinter();

        report.Print(consolePrinter);
        report.Print(pdfPrinter);
    }
}
```

```
interface IPrinter...

class ConsolePrinter ...

class PDFPrinter ...

class HTMLPrinter : IPrinter
{
    public void Print(string text)...
}
```

```
class Report
{
    public string Text { get; set; }

    public void GoToFirstPage()...

    public void GoToLastPage()...

    public void GoToSecondPage()...

    public void GoToPenultimatePage()...

    public void GoToPage(int pageNumber)...

    public void Print(IPrinter printer)...
}
```

Open–closed principle

```
class Cookbook
{
    public string Name { get; set; }

    public void CookDinner()
    {
        Console.WriteLine("Peeling potatoes");
        Console.WriteLine("Cooking potatoes");
        Console.WriteLine("Potatoes are ready");
    }
}
```

```
class Cookbook
{
    public string Name { get; set; }

    public void CookDinner()
    {
        Console.WriteLine("Peeling potatoes");
        Console.WriteLine("Cooking potatoes");
        Console.WriteLine("Potatoes are ready");
    }

    public void CookSalad()
    {
        Console.WriteLine("Preparing vegetables");
        Console.WriteLine("Cut vegetables");
        Console.WriteLine("Vegetables are ready");
    }
}
```

```
interface IMeal
{
    void Make();
}

class PotatoMeal : IMeal
{
    public void Make()...
}
class SaladMeal : IMeal
{
    public void Make()...
}
```

```
public void Make()
{
    Console.WriteLine("Peeling potatoes");
    Console.WriteLine("Cooking potatoes");
    Console.WriteLine("Potatoes are ready");
}
```

```
class Cookbook
{
    public string Name { get; set; }

    public void CookDinner(IMeal meal)
    {
        meal.Make();
    }
}
```

Option 1.

```
abstract class MealBase
{
    public void Make()
    {
        Prepare();
        Cook();
        FinalSteps();
    }
    protected abstract void Prepare();
    protected abstract void Cook();
    protected abstract void FinalSteps();
}
```

```
class PotatoMeal : MealBase
{
    protected override void Cook()
    {
        Console.WriteLine("Cooking potatoes");
    }

    protected override void FinalSteps()
    {
        Console.WriteLine("Potatoes are ready");
    }

    protected override void Prepare()
    {
        Console.WriteLine("Peeling potatoes");
    }
}
```

```
class SaladMeal : MealBase
{
    protected override void Cook()...

    protected override void FinalSteps()...

    protected override void Prepare()...
}
```

Option 2.

```
class Cookbook
{
    public void MakeDinner(MealBase[] menu)
    {
        foreach (MealBase meal in menu)
            meal.Make();
    }
}

abstract class MealBase...

class PotatoMeal : MealB
{
    protected override v

    protected override v

    protected override void Prepare()...
}

class SaladMeal ...
```

```
abstract class MealBase
{
    public void Make()
    {
        Prepare();
        Cook();
        FinalSteps();
    }
    protected abstract void Prepare();
    protected abstract void Cook();
    protected abstract void FinalSteps(
}
```

Option 1.

```
class Program
{
    static void Main(string[] args)
    {
        Cookbook cookbook = new Cookbook();

        cookbook.CookDinner(new PotatoMeal());
        cookbook.CookDinner(new SaladMeal());
    }
}
```

Option 2.

```
class Program
{
    static void Main(string[] args)
    {
        MealBase[] menu = new MealBase[]
            { new PotatoMeal(), new SaladMeal() };

        Cookbook cookbook = new Cookbook();
        cookbook.MakeDinner(menu);
    }
}
```

Liskov substitution principle

```
class Rectangle
{
    public virtual int Width { get; set; }
    public virtual int Height { get; set; }

    public int GetArea()
    {
        return Width * Height;
    }
}
```

```
class Square : Rectangle
{
    public override int Width
    {
        get
        {
            return base.Width;
        }

        set
        {
            base.Width = value;
            base.Height = value;
        }
    }

    public override int Height...
}
```

```
class Rectangle
{
    public virtual int Width { get; set; }
    public virtual int Height { get; set; }

    public int GetArea()
    {
        return Width * Height;
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Rectangle rect = new Square();
        TestRectangleArea(rect);

        Console.Read();
    }

    public static void TestRectangleArea(Rectangle rect)
    {
        rect.Height = 5;
        rect.Width = 10;
        if (rect.GetArea() != 50)
            throw new Exception("Incorrect area!");
    }
}
```

```
static void Main(string[] args)
{
    Rectangle rect = new Square();
    TestRectangleArea(rect);

    Console.Read();
}

public static void TestRectangleArea(Rectangle rect)
{
    if (rect is Square)
    {
        rect.Height = 5;
        if (rect.GetArea() != 25)
            throw new Exception("Incorrect area!");
    }
    else if (rect is Rectangle)
    {
        rect.Height = 5;
        rect.Width = 10;
        if (rect.GetArea() != 50)
            throw new Exception("Incorrect area!");
    }
}
```

Interface segregation principle

```
interface IMessage
{
    void Send();
    string Text { get; set; }
    string Subject { get; set; }
    string ToAddress { get; set; }
    string FromAddress { get; set; }
}

class EmailMessage : IMessage
{
    public string Subject { get; set; }
    public string Text { get; set; }
    public string FromAddress { get; set; }
    public string ToAddress { get; set; }

    public void Send()
    {
        Console.WriteLine("Send Email: {0}", Text);
    }
}
```

```
class TextMessage : IMessage
{
    public string Text { get; set; }
    public string FromAddress { get; set; }
    public string ToAddress { get; set; }

    public string Subject
    {
        get
        {
            throw new NotImplementedException();
        }

        set
        {
            throw new NotImplementedException();
        }
    }

    public void Send()
    {
        Console.WriteLine("Send Sms: {0}", Text);
    }
}
```

```
interface IMessage
{
    void Send();
    string Text { get; set; }
    string Subject { get; set; }
    string ToAddress { get; set; }
    string FromAddress { get; set; }
}
```

```
interface IMessage
{
    void Send();
    string Text { get; set; }
    string ToAddress { get; set; }
    string Subject { get; set; }
    string FromAddress { get; set; }

    byte[] Voice { get; set; }
}
```

```
class VoiceMessage : IMessage
{
    public string ToAddress { get; set; }
    public string FromAddress { get; set; }
    public byte[] Voice { get; set; }

    public string Text { get => throw new NotImplementedException(); set => throw new NotImplementedException(); }
    public string Subject { get => throw new NotImplementedException(); set => throw new NotImplementedException(); }

    public void Send() { Console.WriteLine("Send voice"); }
}

class EmailMessage : IMessage
{
    public string Subject { get; set; }
    public string Text { get; set; }
    public string FromAddress { get; set; }
    public string ToAddress { get; set; }

    public byte[] Voice { get => throw new NotImplementedException(); set => throw new NotImplementedException(); }

    public void Send() { Console.WriteLine("Send Email: {0}", Text); }
}
```

```
interface IMessage
{
    void Send();
    string Text { get; set; }
    string ToAddress { get; set; }
    string Subject { get; set; }
    string FromAddress { get; set; }

    byte[] Voice { get; set; }
}
```

```
interface IMessage
{
    void Send();
    string ToAddress { get; set; }
    string FromAddress { get; set; }
}
interface IVoiceMessage : IMessage
{
    byte[] Voice { get; set; }
}
interface ITextMessage : IMessage
{
    string Text { get; set; }
}

interface IEmailMessage : ITextMessage
{
    string Subject { get; set; }
}
```

```
class VoiceMessage : IVoiceMessage
{
    public string ToAddress { get; set; }
    public string FromAddress { get; set; }

    public byte[] Voice { get; set; }
    public void Send()...
}
class EmailMessage : IEmailMessage
{
    public string Text { get; set; }
    public string Subject { get; set; }
    public string FromAddress { get; set; }
    public string ToAddress { get; set; }

    public void Send()...
}

class TextMessage : ITextMessage
{
    public string Text { get; set; }
    public string FromAddress { get; set; }
    public string ToAddress { get; set; }

    public void Send()...
}
```

Dependency inversion principle

```
class Book
{
    public string Text { get; set; }
    public ConsolePrinter Printer { get; set; }

    public void Print()
    {
        Printer.Print(Text);
    }
}

class ConsolePrinter
{
    public void Print(string text)
    {
        Console.WriteLine(text);
    }
}
```

```
class Book
{
    public string Text { get; set; }
    public HtmlPrinter Printer { get; set; }

    public void Print()
    {
        Printer.Print(Text);
    }
}

class ConsolePrinter
{
    public void Print(string text)...
}

class HtmlPrinter
{
    public void Print(string text)...
}
```

```
interface IPrinter
{
    void Print(string text);
}

class Book
{
    public string Text { get; set; }
    public IPrinter Printer { get; set; }

    public Book(IPrinter printer) { this.Printer = printer; }

    public void Print() { Printer.Print(Text); }
}

class ConsolePrinter : IPrinter
{
    public void Print(string text)...
}

class HtmlPrinter : IPrinter
{
    public void Print(string text)...
}
```

```
static void Main(string[] args)
{
    Book book = new Book(new ConsolePrinter());
    book.Print();

    book.Printer = new HtmlPrinter();
    book.Print();
}
```

Except for SOLID principles, there are also other principles:

- **KISS** — Keep It Simple, Stupid!
- **DRY** — Don’t Repeat Yourself
- **YAGNI** — You Ain’t Gonna Need It
- **GRASP** — General responsibility assignment software patterns
- **GoF** — Gang of Four