Clean Code:
A Reader-Centered Approach

@matthewrenze

#sddconf

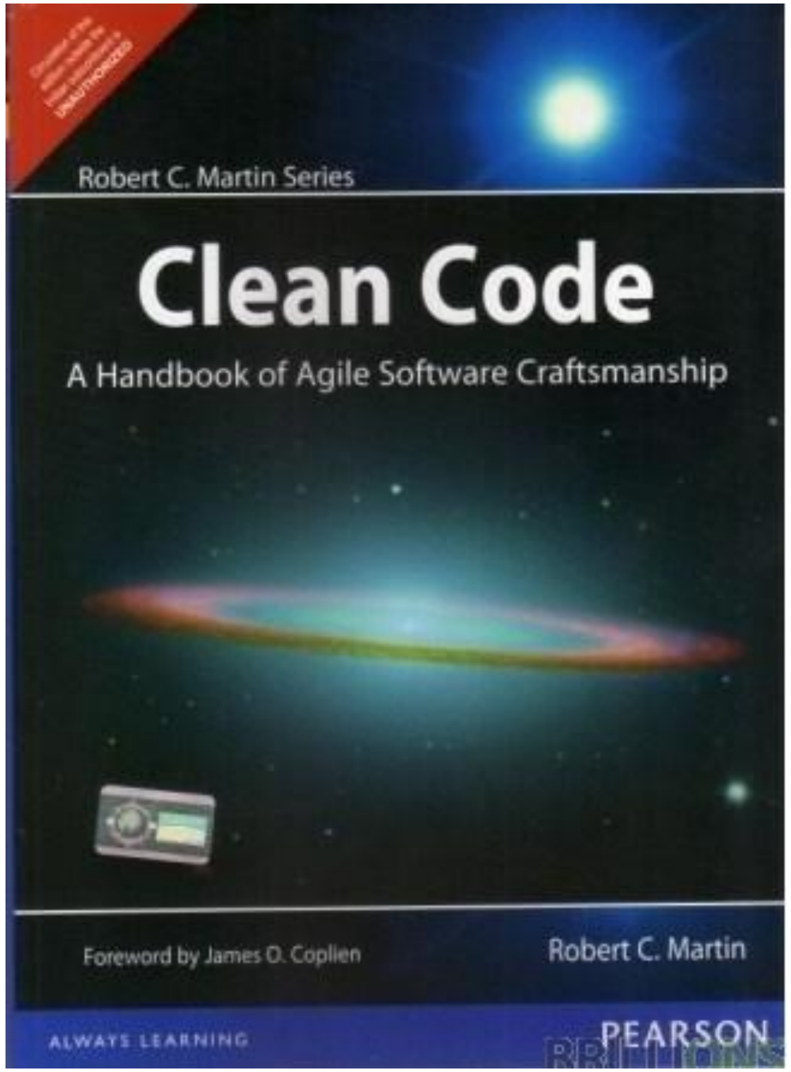
FIRST
AID
STAR-MEDICAL, INC
1) 100% DOOR COVERAGE/ CCTV
2) ASSET PROTECTION
3) RECEIPT CHECKING
4) TALKING SHRINK
READ AND FOLLOW

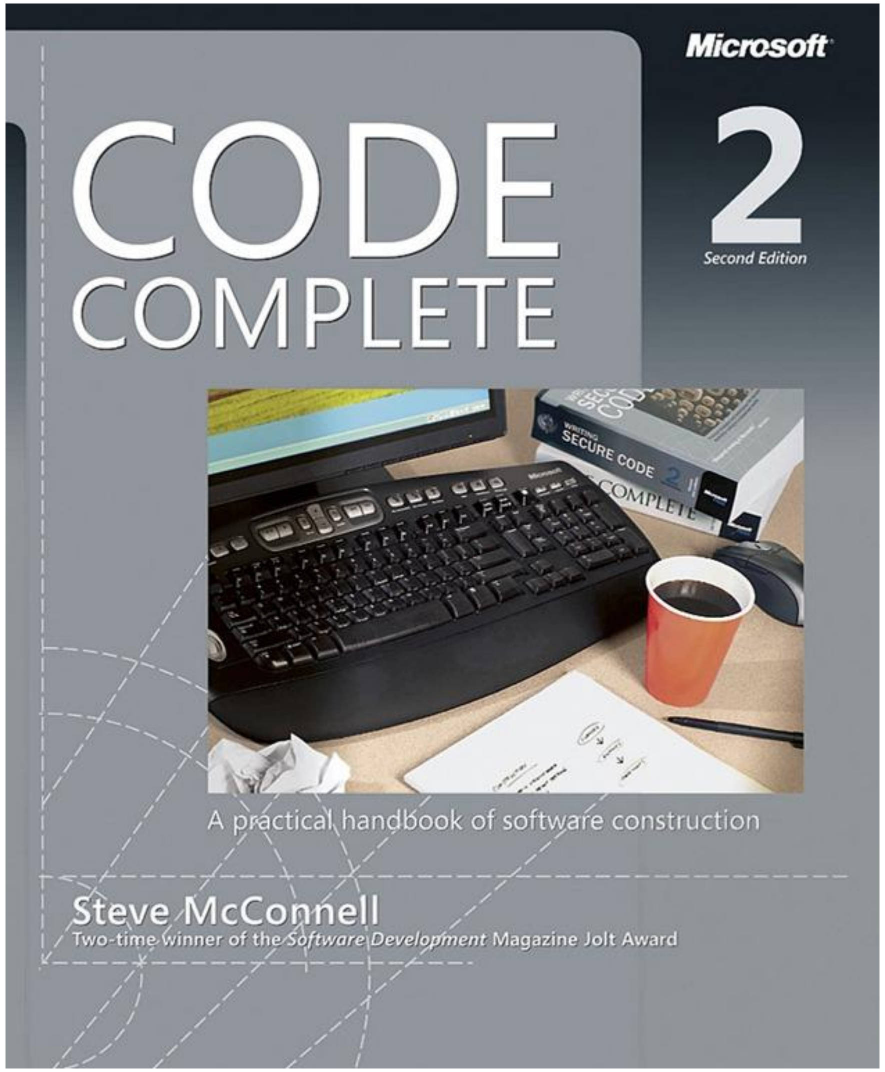
TECHNICAL COMMUNICATION

[A READER-CENTERED APPROACH]

SIXTH EDITION

PAUL V. ANDERSON


Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin
ALWAYS LEARNING
PEARSON


Microsoft
CODE COMPLETE
2
Second Edition
A practical handbook of software construction
Steve McConnell
Two-time winner of the Software Development Magazine Jolt Award

About Me

Independent consultant

Education

- B.S. in Computer Science (ISU)
- B.A. in Philosophy (ISU)

Community

- Public Speaker
- Pluralsight Author
- Microsoft MVP
- ASPInsider
- Open-Source Software

Overview

Clean Code

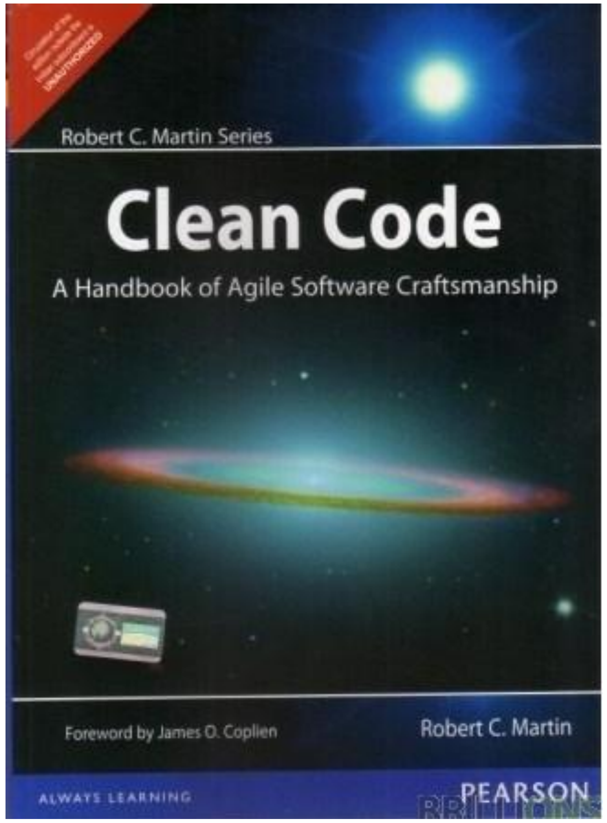
Names

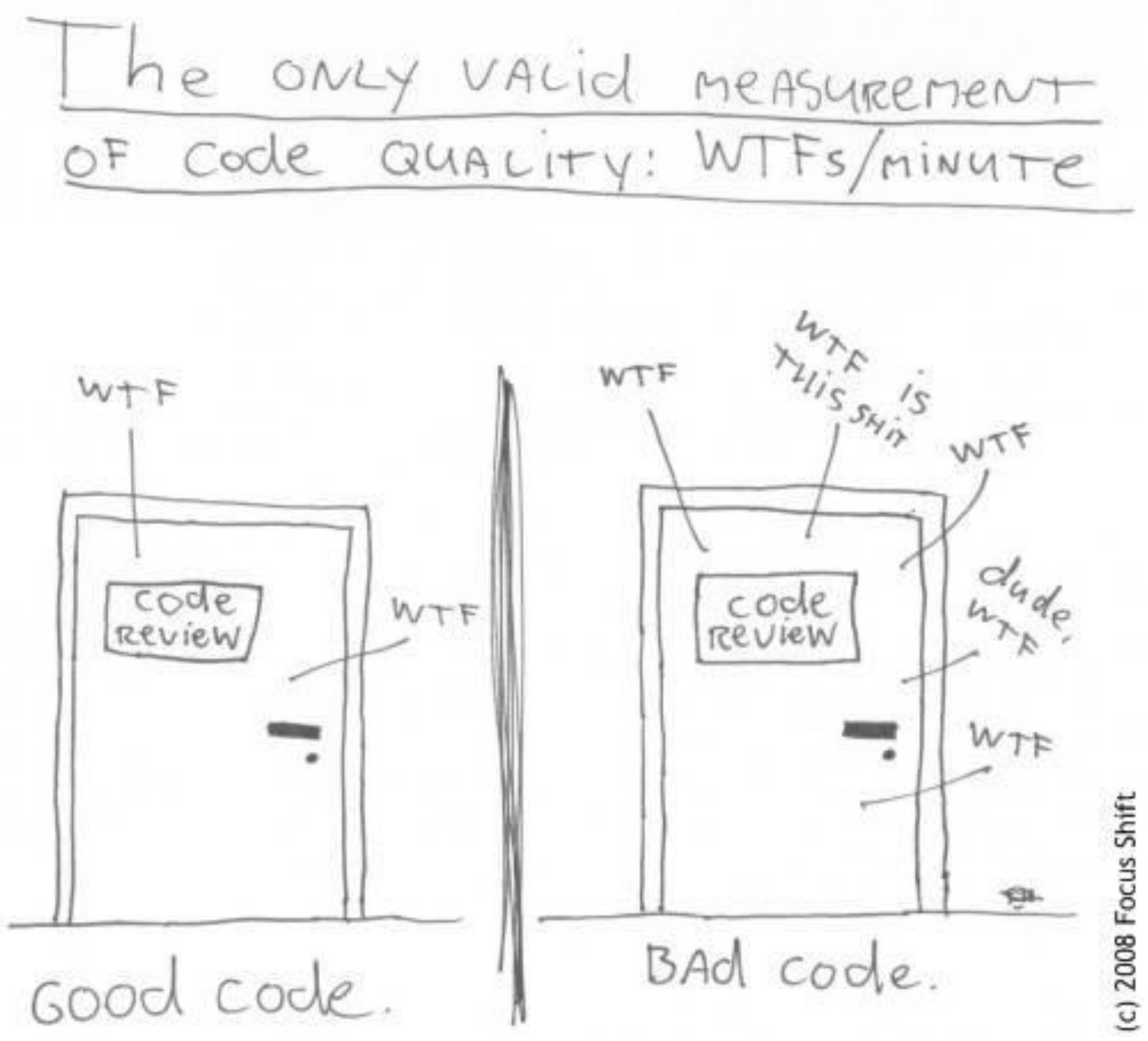
Functions

Classes

Comments

Process





Source: http://www.bernie-eng.com/blog/wp-content/uploads/2011/03/code.jpg

What is Bad Code?

Difficult to read

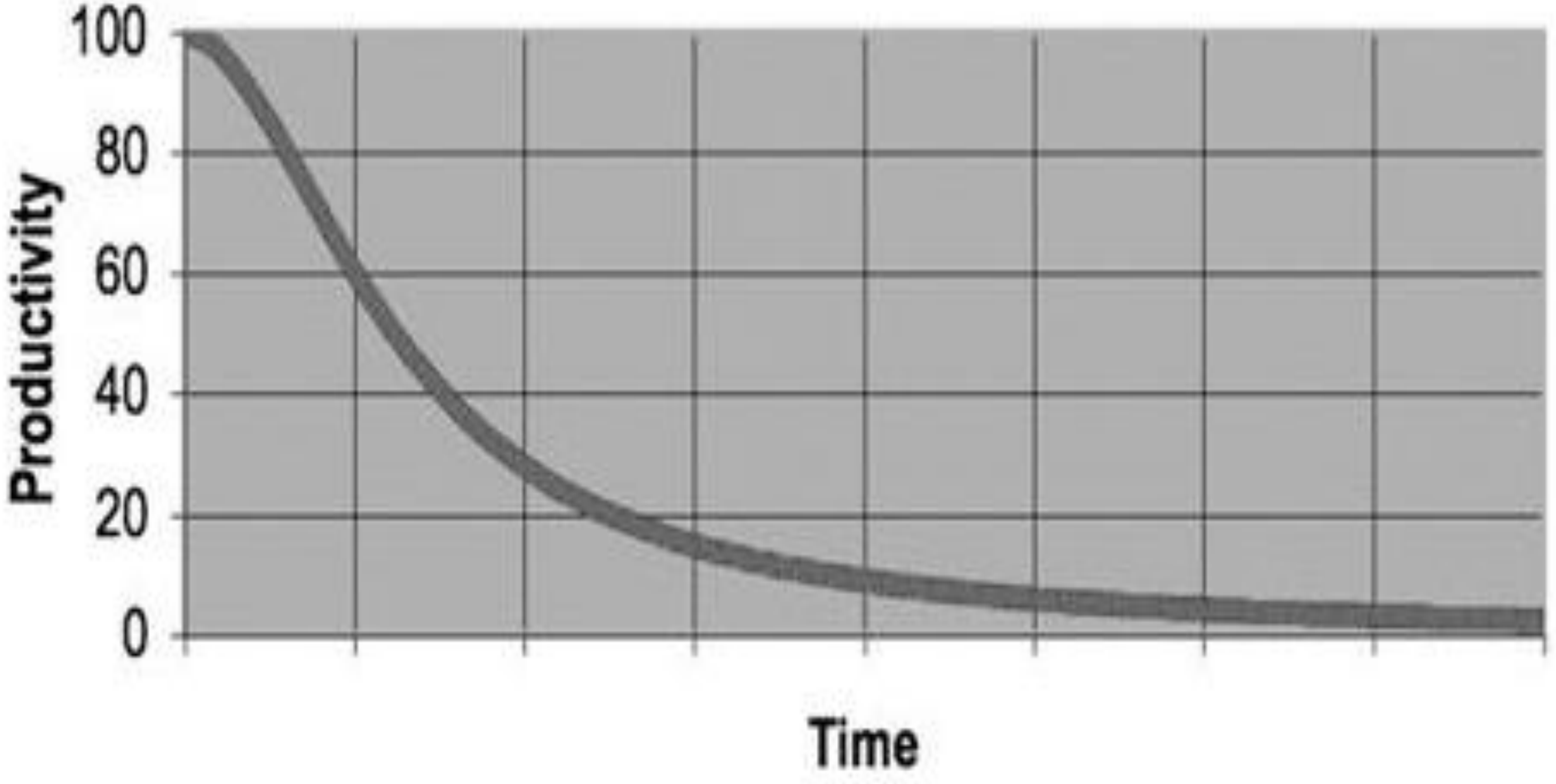
Difficult to understand

Difficult to maintain

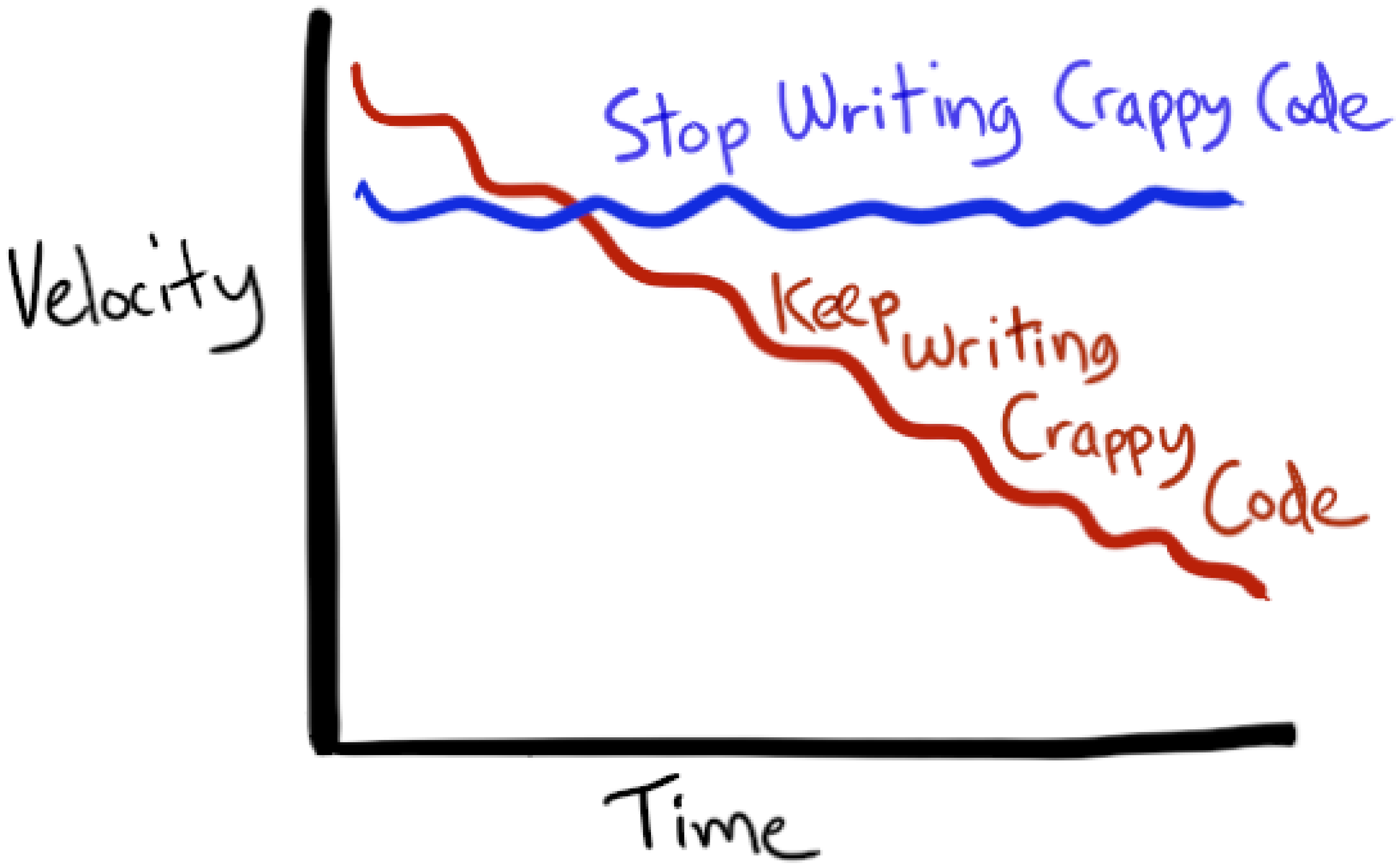
Contains bugs

Contains surprises

The Total Cost of Owning a Mess



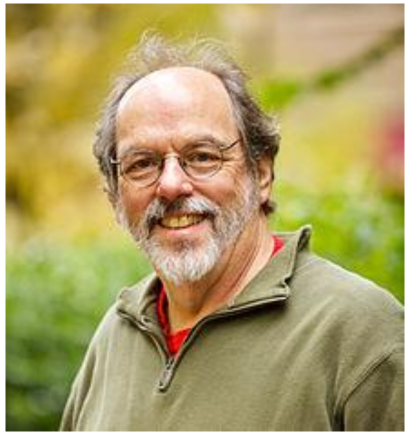
Source: Clean Code



Source: http://talkingincode.com/wp-content/uploads/2014/12/crapcode.png

The way we avoid a mess is by keeping our code clean.

What is Clean Code?



What is Clean Code?

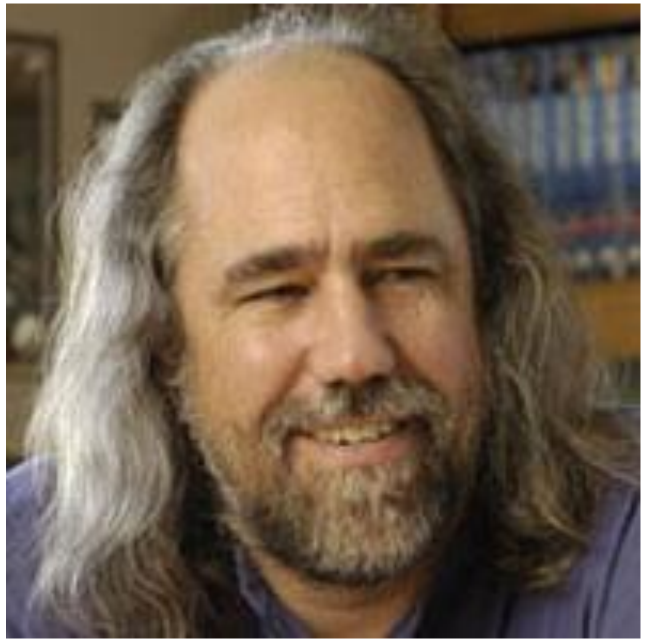
Simple and direct

Reads like well-written prose

Never obscures the designer’s intent

Full of crisp abstractions

Contains straight-forward lines of control



Grady Booch

Co-inventor of UML

What is Clean Code?

Runs all the tests

Expresses all the design ideas in the system

Minimizes the number of entities

Minimizes duplication

Expresses ideas clearly

Ron Jeffries

Co-inventor of XP

What is Clean Code?

Readable by others

Has unit tests

Has meaningful names

Has minimal dependencies

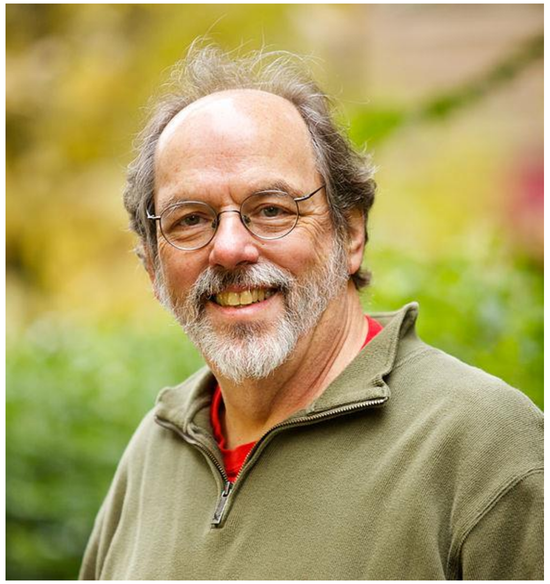
Do one thing

Dave Thomas

Co-Author of
The Pragmatic Programmer

What is Clean Code?

"You know you are working on clean code when each routine you read turns out to be pretty much what you expected."



Ward Cunningham

Inventor of the Wiki
Co-inventor of XP

What is Clean Code?

Simple

Readable

Understandable

Maintainable

Testable

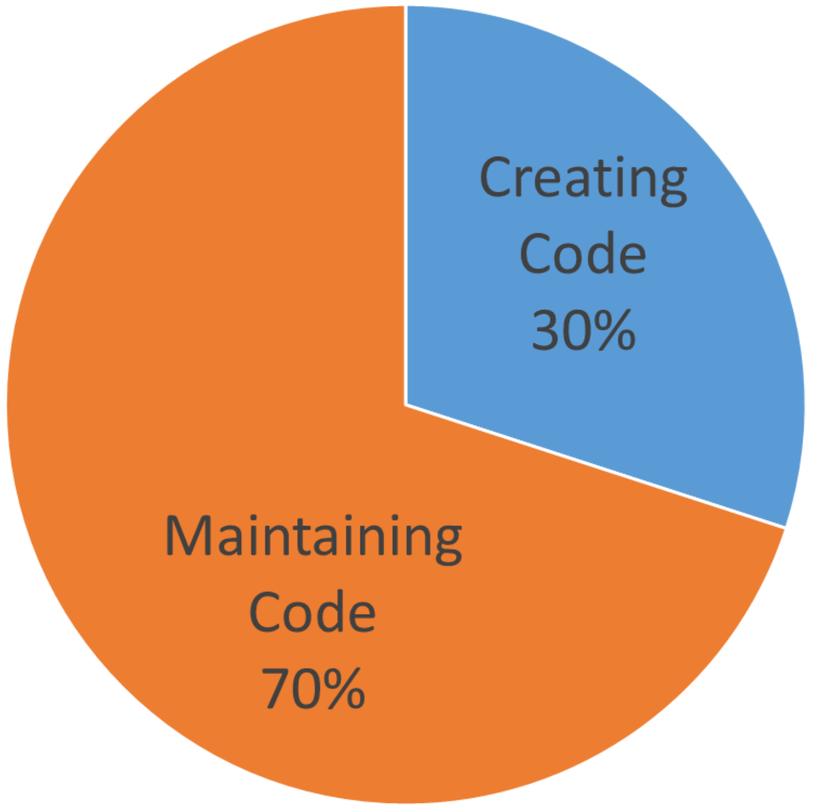
Matthew Renze

Not really famous for anything… yet :)

What is Clean Code?

Code that is written for the **reader** of the code… not for the author… or the machine

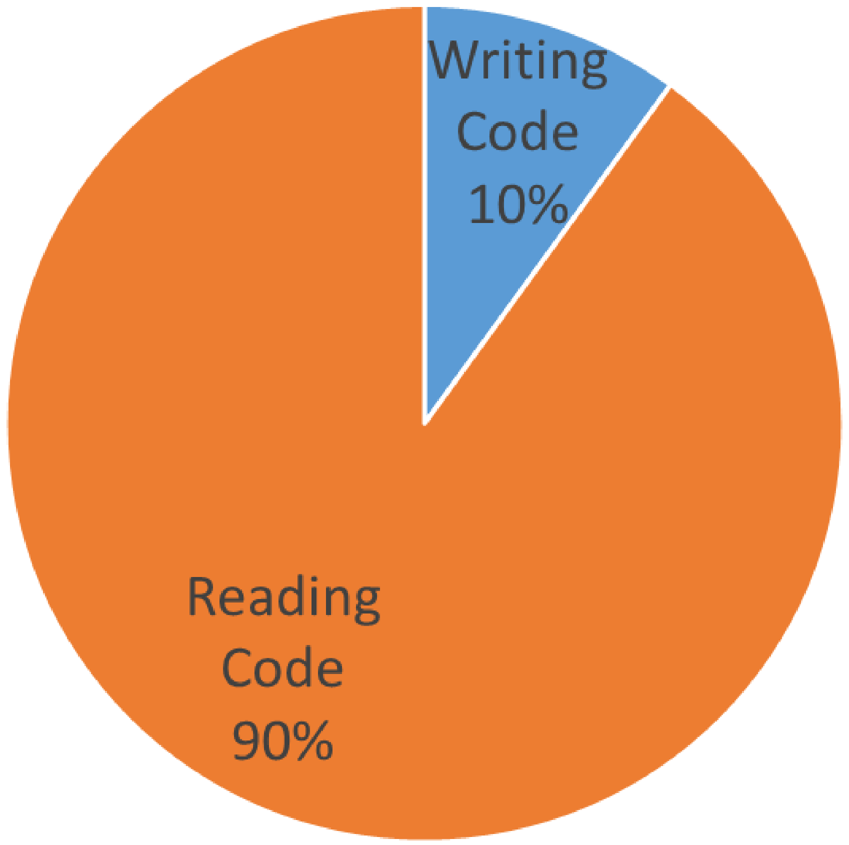
Why Should We Invest in Clean Code?



Sources:

- Barry Boehm - Software Engineering Economics, Prentice Hall
- Schach, R., Software Engineering, Fourth Edition, McGraw-Hill
- Glass, Robert, Frequently Forgotten Fundamental Facts about Software Engineering

Why Should We Invest in Clean Code?



Source: Clean Code

Clean Code is an Investment

Clean code makes it easier to:

- Write new code
- Maintain old code

Invest in code readability

How Do You Write Clean Code?

Write code for the *reader*

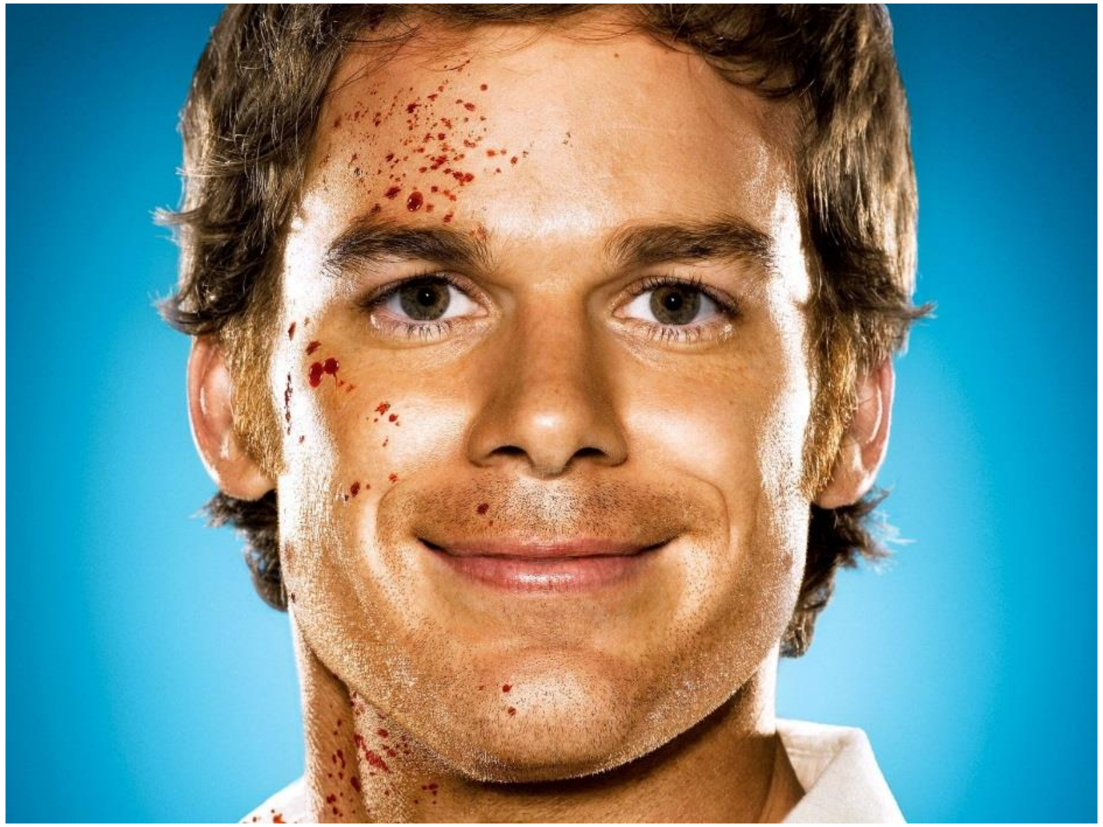
Not for the *author*

Not for a *machine*

How Do You Write Clean Code?

"Always code as if the person who ends up maintaining your code is a violent psychopath who knows where you live!"

- Author Unknown



Names

Choose Names Thoughtfully

Use Intention-Revealing Names

```
// Bad - Terse variable name
int d;  // days in queue

// Good
int daysInQueue;
```

Use Intention-Revealing Names

```
// Bad – Unclear method name
private int Process();

// Good
private int ParseCustomerIdFromFile();
```

Use Names from Problem Domain

```
// Problem domain
public class Customer {}

public void AddAccount();
```

Use Names from Solution Domain

```
// Solution domain
public class Factory {}

public void AddToJobQueue();
```

Use Names from Both Domains

```
// Both domains
public class CustomerFactory {}

public void AddAccountToJobQueue();
```

Avoid Disinformation

```
// Bad - misleading
ISet<Customer> customerList;
```

Use Pronounceable Names

```
// Bad – Not pronounceable names
public class DtaRcrd102
{
    private DateTime genymdhms;

    private DateTime modymdhms;

    private string pszqint = "102";
}
```

Use Pronounceable Names

```
// Bad – Not pronounceable names
public class DtaRcrd102
{
    private DateTime genymdhms;

    private DateTime modymdhms;

    private string pszqint = "102";
}
```

```
// Good – Pronounceable names
public class Customer
{
    private DateTime generationTimestamp;

    private DateTime modificationTimestamp;

    private string recordId = "102";
}
```

Avoid Encodings

```
// Bad – Hungarian Notation
private int intSomeValue = 123;
```

Avoid Encodings

```
// Bad - Module prefixes
private int m_SomeField = 0;
```

Avoid Encodings

```
// OK... Maybe?
private int _someField = 0;
```

Class Names

```
// Good - Noun or noun phrase
public class Customer

public class AddressParser

public class AddAccountCommand
```

Class Names

```
// Good - Noun or noun phrase
public class Customer

public class AddressParser

public class AddAccountCommand
```

```
// Bad - Fuzzy names
public class ObjectManager

public class EntityProcessor

public class Stuff
```

Method Names

```
// Good - Verb or verb phrase
public void AddCustomer()

public void DeleteAccount()

public string ParseAddress()
```

Method Names

```
// Good - Verb or verb phrase
public void AddCustomer()

public void DeleteAccount()

public string ParseAddress()
```

```
// Bad - Fuzzy names
public string Process()

public void DoWork()
```

Method Names

```
// Good - Boolean predicates
public bool IsValid()

public bool HasAccount()
```

Length of Variable Names Should Increase with Scope

```
// Good - Very short range variable names
for (int i = 0; i < 10; i++) {}

list.Sum(p => p.GetAmount());
```

Length of Variable Names Should Increase with Scope

```
// Good - Very short range variable names
for (int i = 0; i < 10; i++) {}

list.Sum(p => p.GetAmount());

// Good - Short method variable names
var balance = GetAccountBalance();
```

Length of Variable Names Should Increase with Scope

```
// Good - Very short range variable names
for (int i = 0; i < 10; i++) {}

list.Sum(p => p.GetAmount());

// Good - Short method variable names
var balance = GetAccountBalance();

// Good - Longer field variable names
private int totalAccountBalance = 0;
```

Length of Variable Names Should Increase with Scope

```
// Good - Very short range variable names
for (int i = 0; i < 10; i++) {}

list.Sum(p => p.GetAmount());

// Good - Short method variable names
var balance = GetAccountBalance();

// Good - Longer field variable names
private int totalAccountBalance = 0;

// Good - Even longer global variable names
global int totalBalanceInAllBankAccounts;
```

Length of Method Names Should Decrease with Scope

```
// Good - Short public method names
public void GetCustomers();

public void Save();
```

Length of Method Names Should Decrease with Scope

```
// Good - Short public method names
public void GetCustomers();

public void Save();


// Good - Longer private method names
private string ParseHtmlFromFile()

private int GetIdFromAccountHolder()
```

Length of Class Names Should Decrease with Scope

```
// Good - Short public class name
public class Account
```

Length of Class Names Should Decrease with Scope

```
// Good - Short public class name
public class Account


// Good - Longer private class name
private class AccountNumberGenerator
```

Length of Class Names Should Decrease with Scope

```
// Good - Short public class name
public class Account


// Good - Longer private class name
private class AccountNumberGenerator


// Good - Longer derived class name
public abstract class Account

public class SavingsAccount : Account
```

Functions

Functions Should Be Small

Simpler

Easier to read

Easier to understand

Easier to test

Contain less bugs

How Small?

Most evidence says:

Less than 20 lines

Uncle Bob says:

Less than 10 lines

Average 3 to 6 lines

Large Functions are Where Classes Go to Hide

Functions Should Do One Thing

Source: http://www.wengerna.com/giant-knife-16999

One Level of Abstraction per Function

```
// Good - Separate levels of abstraction
public File CreateFile()

public Html RenderHtml()

private string RenderHtmlBody()

private string RenderHtmlElement()

private char RenderHtmlElementClosingTag()
```

Minimize the Number of Parameters

```
// Try to minimize the # of arguments
public void SetNone() {}

public void SetOne(int arg1)

public void SetTwo(int arg1, int arg2)

public void SetThree(int arg1, int arg2, int arg3)

public void SetMany(Args args)
```

Avoid Flag Arguments

```
// Bad – Flag arguments
public void Render(bool useColor)
```

Avoid Flag Arguments

```
// Bad – Flag arguments
public void Render(bool useColor)

// Good – No flag arguments
public void RenderInColor()

public void RenderInGrayScale()
```

Avoid Output Arguments

```
// Bad - Uses ‘out’ argument
public void AppendFooter(out Report report)
{
  …
}

AppendFooter(out report);
```

Avoid Output Arguments

```
// Bad - Uses ‘out’ argument
public void AppendFooter(out Report report)
{
  …
}

AppendFooter(out report);


// Good - No ‘out’ argument
public ReportBuilder AppendFooter()
{
  …
}

reportBuilder.AppendFooter();
```

Command-Query Separation

Command

Does something

Should modify state

Should not return a value

Command-Query Separation

Command

Does something

Should modify state

Should not return a value

Query

Answers a question

Should not modify state

Always returns a value

Command-Query Separation

Command

Does something

Should modify state

Should not return a value

Query

Answers a question

Should not modify state

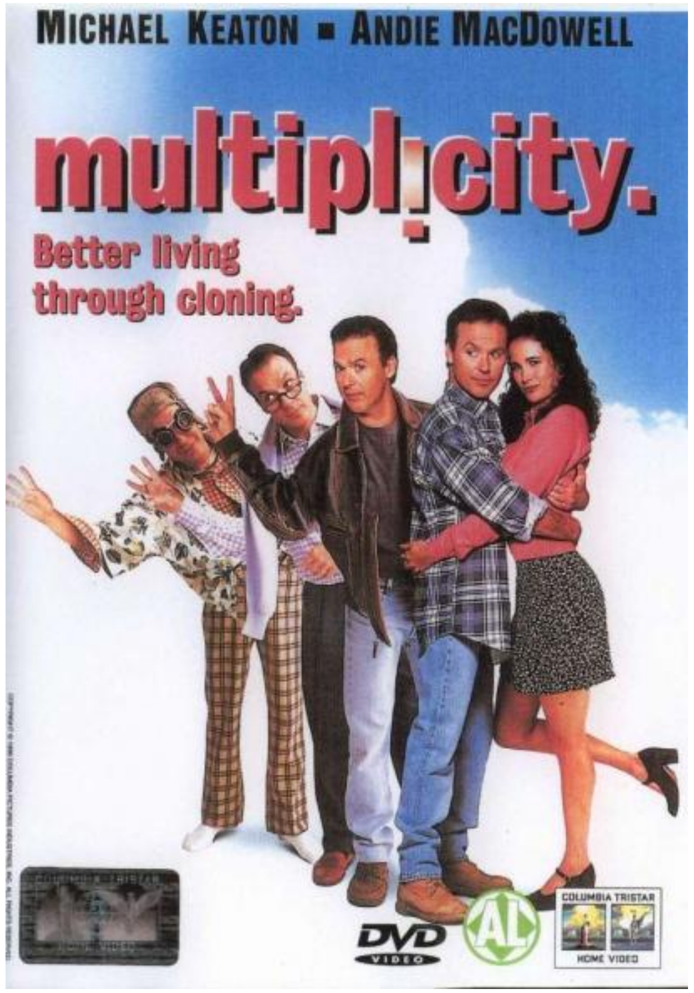
Always returns a value

Avoid mixing the two!

Avoid Side Effects

May cause
Dizziness
May cause
headache
DO NOT take with
Nitrates.
Some medicines can change
the effect of this medicine. Check
with your doctor or pharmacist
before taking any other medicine

Avoid Duplication


MICHAEL KEATON ▪ ANDIE MACDOWELL
multiplicity.
Better living through cloning.
DVD VIDEO
AL
COLUMBIA TRISTAR HOME VIDEO

Source: Sony Pictures Home Entertainment

Use Functions to Enhance Readability

```
// Bad – One giant chunk of code
public void CreateReport()
{
 ... Giant block of code ...
}
```

Use Functions to Enhance Readability

```
// Bad – One giant chunk of code
public void CreateReport()
{
 ... Giant block of code ...
}

// Good – Uses small named functions
public void CreateReport()
{
    CreateHeader();

    CreateBody();

    CreateFooter();
}
```

Classes

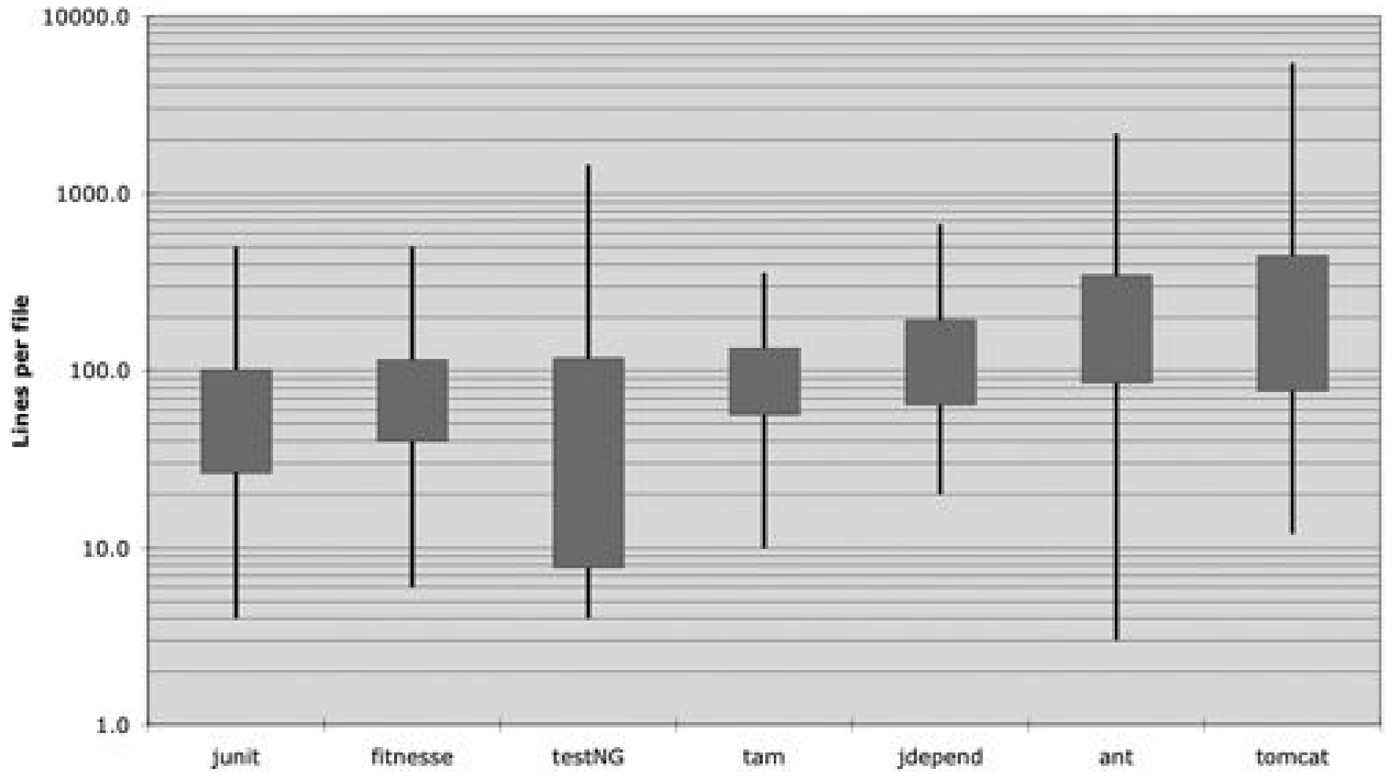
Classes Should Be Small

Similar benefits as small functions

Single-Responsibility Principle

How Small?

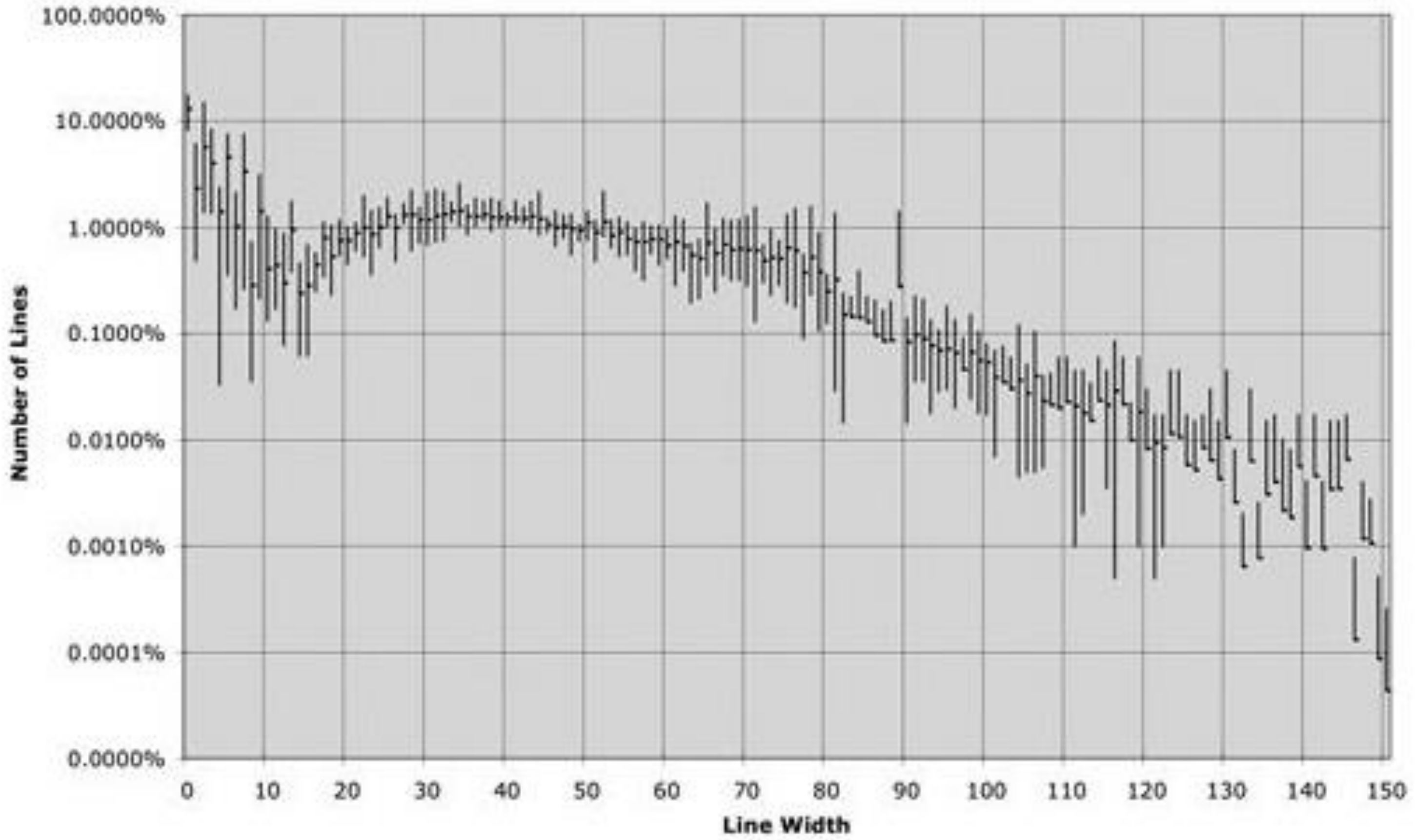
Figure 5-1 File length distributions LOG scale (box height = sigma)



Source: Clean Code

Classes Should Be Narrow

Figure 5-2 Java line width distribution



Source: Clean Code

Follow the Law of Demeter

```
// Bad - Law of Demeter violation
var rent = customer.Pocket.Wallet
     .Money.GetRentMoney();
```

Follow the Law of Demeter

```
// Bad - Law of Demeter violation
var rent = customer.Pocket.Wallet
     .Money.GetRentMoney();

// Good - No violation
var rent = customer.GetRentMoney();
```

Follow the Law of Demeter

```
// Bad - Law of Demeter violation
var rent = customer
     .Pocket.Wallet
     .Money.GetRentMoney();

// Good - No violation
var rent = customer.GetRentMoney();
```

Source: Athens Banner-Herald

Object vs. Data Structure

```
public class Rectangle
{
    private double x;
    ...
    public double GetX()
    {
        return x;
    }
    ...
    public double GetArea()
    {
        return width * height;
    }
}
```

Object vs. Data Structure

```
public class Rectangle
{
    private double x;
    ...
    public double GetX()
    {
        return x;
    }
    ...
    public double GetArea()
    {
        return width * height;
    }
}
```

```
public struct Rectangle
{
    public double X;

    public double Y;

    public double Width;

    public double Height;
}
```

Avoid Hybrid Object/Structures

Source: http://www.layoutsparks.com/1/147428
/alien-resurrection-scary-dreadful-31000.html

Have a Consistent Order

```
public class SomeClass
{
    private const int SomeConst = 123;

    private int _someField;

    private int SomeProperty {...}

    public SomeClass() {...}

    public void DoSomethingPublic() {...}

    private void DoSomethingPrivate() {...}
}
```

Choose the Right Abstractions

Model

View

Controller

Repository

Factory

Builder

Adapter

Other Practices for Classes

DRY Principle

High Cohesion

Low Coupling

Dependency Injection

Testability

Comments

Comments Represent a Failure

Source: http://a.tgcdn.net/images/products/zoom/no_comment.jpg

Obsolete Comments Lie

Explain Yourself in Code

```
// Bad - Code explained in comment
// Check to see if the employee is eligible for full benefits
if ((employee.FullTime || SalaryFlag)
    && (employee.Age > 65))
```

Explain Yourself in Code

```
// Bad - Code explained in comment
// Check to see if the employee is eligible for full benefits
if ((employee.FullTime || SalaryFlag)
    && (employee.Age > 65))

// Good - Code explains itself
private bool IsEligibleForFullBenefits(Employee employee)
{
    return ((employee.FullTime || SalaryFlag)
        && employee.Age > 65))
}
```

Explain Yourself in Code

```
// Bad - Code explained in comment
// Check to see if the employee is eligible for full benefits
if ((employee.FullTime || SalaryFlag)
    && (employee.Age > 65))

// Good - Code explains itself
private bool IsEligibleForFullBenefits(Employee employee)
{
    return ((employee.FullTime || SalaryFlag)
        && employee.Age > 65))
}

if (IsEligibleForFullBenefits(employee))
```

Bad Comments

```
// All of these comments are bad

// Opens the file
var file = File.Open();

// Returns day of month
private int GetDayOfWeek()

// 08-07-2013 – Fixed Bug (MLR)

Main()
{
  …
} // end main
```

Zombie Code

```
// Zombie Code
// if (a == 1)
//     b = c + 1
```

Source: The Walking Dead

Zombie Code

```
// Zombie Code
// if (a == 1)
//     b = c + 1
```

Kill it with fire!

Source: The Walking Dead

Necessary Comments

```
// Copyright © 2017 Matthew Renze

// Trim is necessary to prevent a
// search term mismatch

// Warning: Slow running test

// TODO: Refactor to factory pattern

/// <summary>
/// Opens the file for reading
/// </summary>
```

The Best Comment is No Comment at All

(but only if our code clearly explains itself)

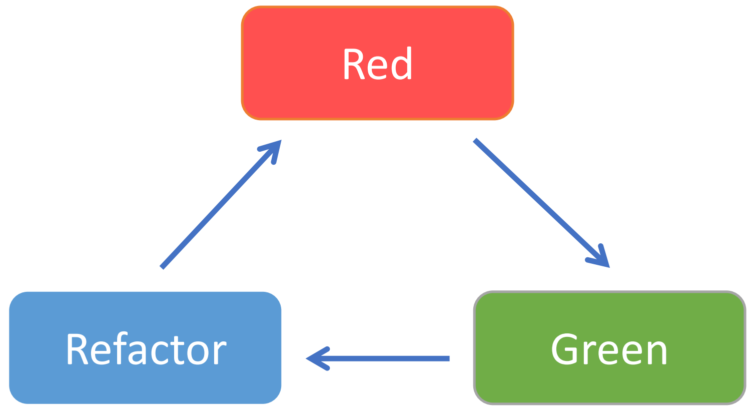
The Process

The Principles

Test-Driven Development (TDD)

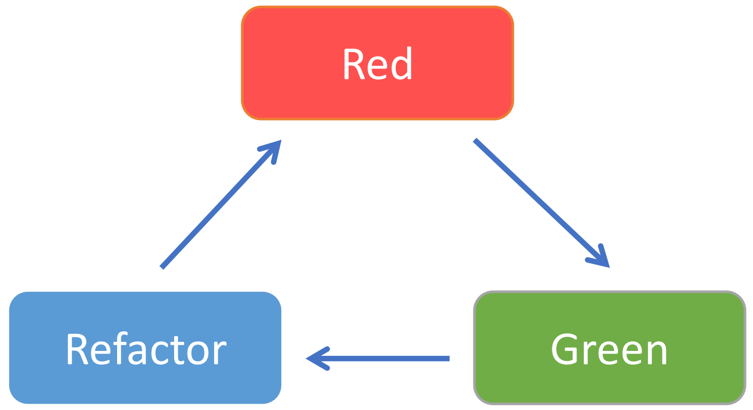
Simplicity (KISS)

Continuous Refactoring



Test-Driven Development Process

1. Create a failing unit test
2. Code the simplest thing
3. Refactor until the code is clean

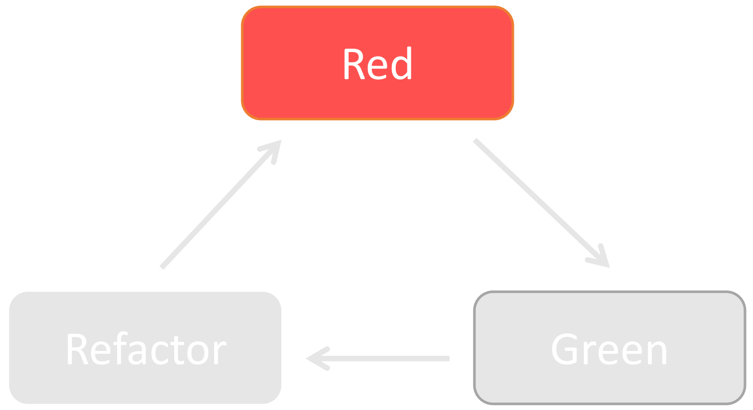


Test-Driven Development

Starts with a test

Tests drive the design

Code evolves over time



TDD Code is:

Testable

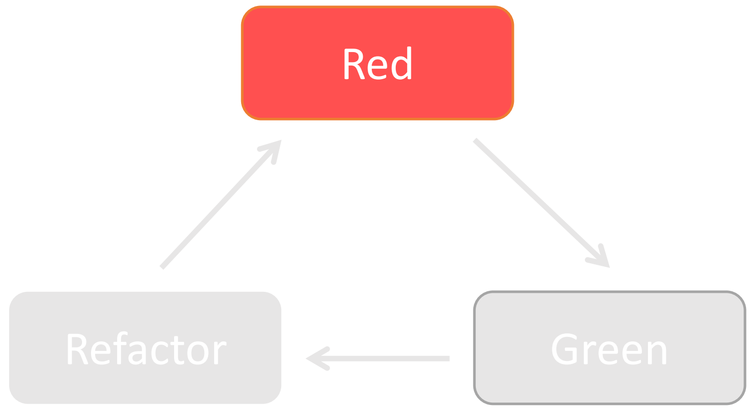
Maintainable

Reliable

Self-documenting

Clean

Easy to keep clean



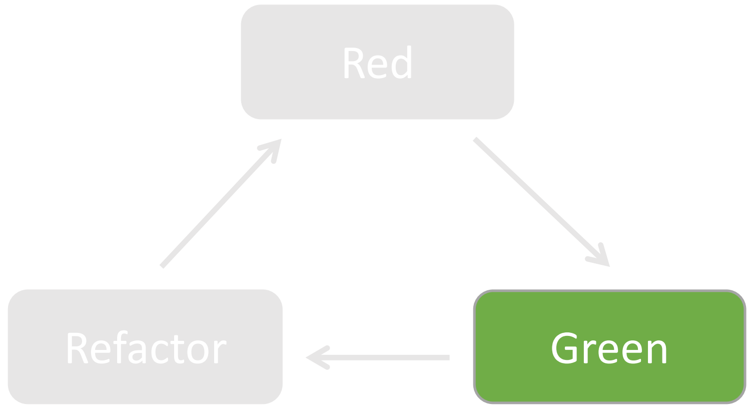
Simplicity

Keep it Simple (KISS)

Unnecessary complexity

You Ain’t Gonna Need It (YAGNI)

Incremental algorithmics

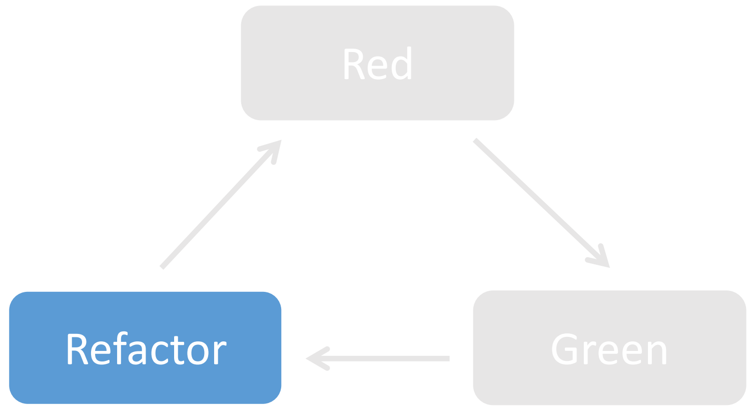


Continuous Refactoring

Working code is *not* the last step

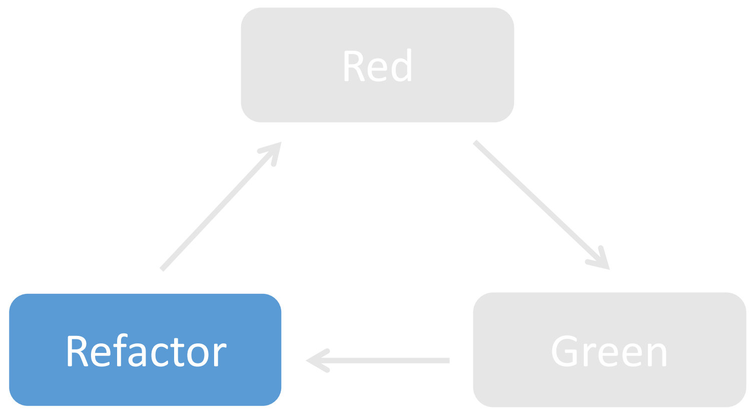
Refactor until clean

Continuous process



Continuous Refactoring

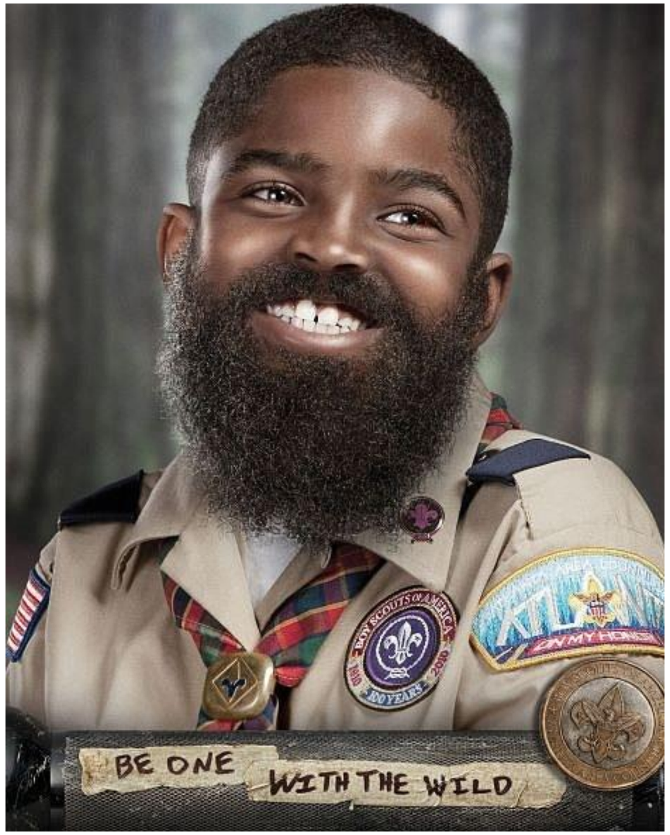
All creative endeavors are iterative processes



Follow the Boy Scout Rule

“Leave the campground just a little bit cleaner than you found it.”

– adapted from Robert Stephenson Smyth Baden-Powell’s farewell message to the scouts: “*Try and leave this world a little better than you found it.*”



"Leave the campground just a little bit cleaner than you found it."

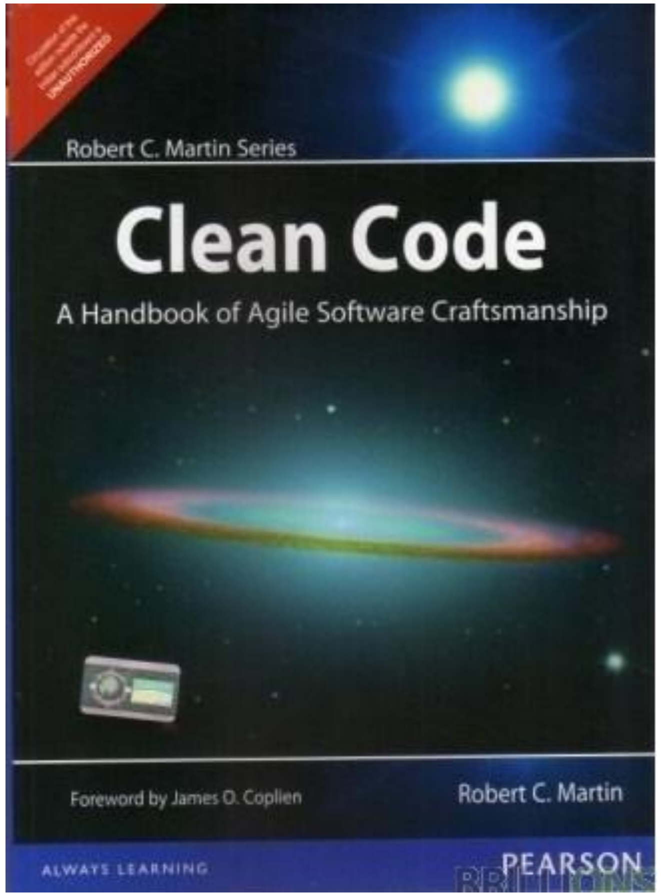
Conclusion

Conclusion

Clean code is:

- Simple
- Readable
- Understandable
- Maintainable
- Testable

Clean code is a philosophy of writing code for the reader



Conclusion

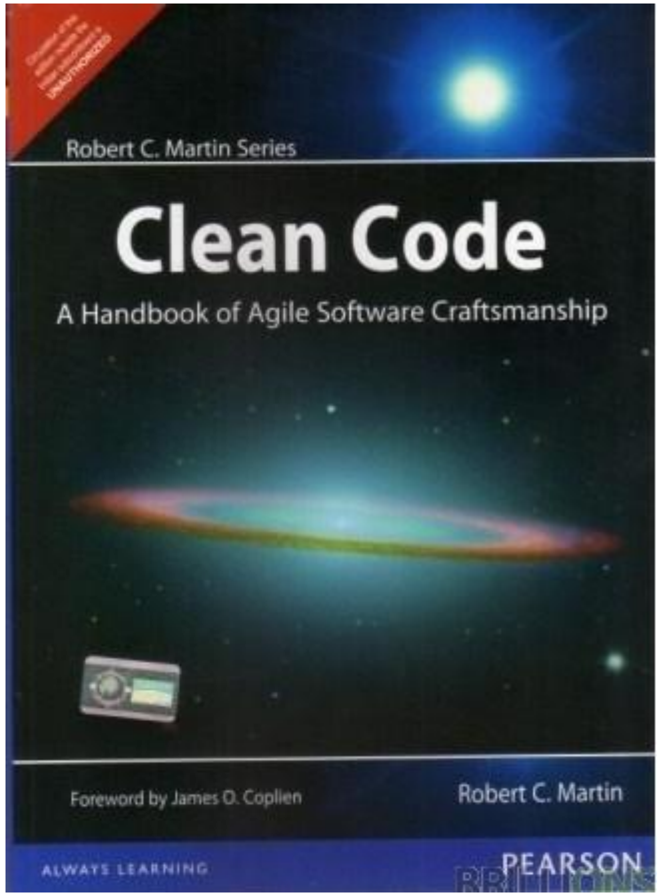
Use intention revealing names

Classes and functions should be small

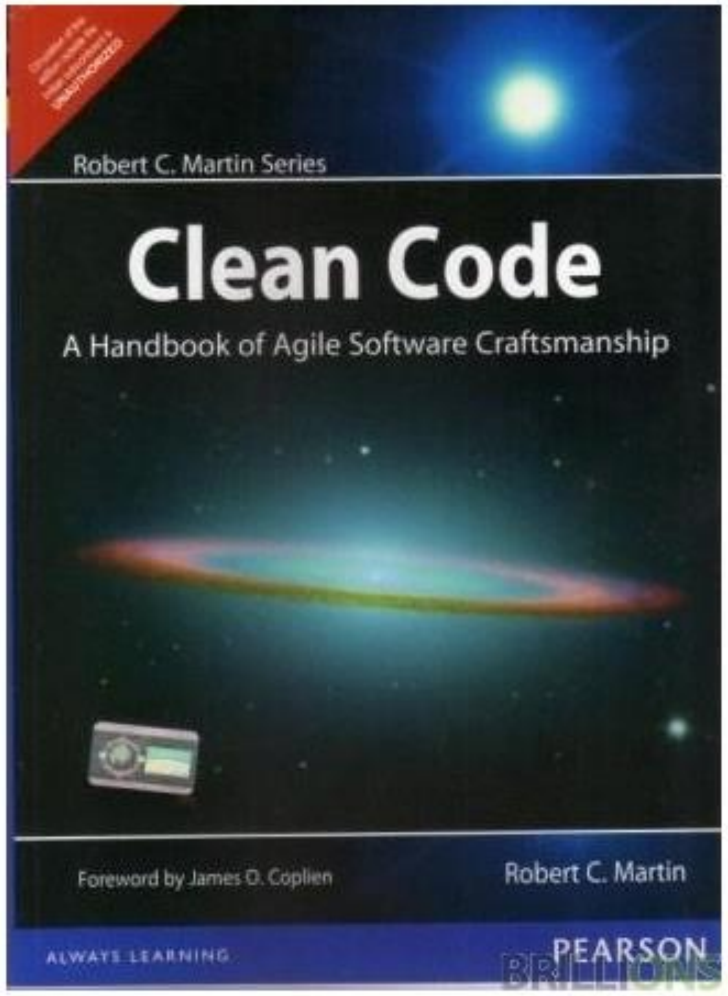
Use comments to express a failure

The process is:

1. Test First (TDD)
2. Simplest solution
3. Continuously refactor



Where to Go Next


Robert C. Martin Series

Clean Code

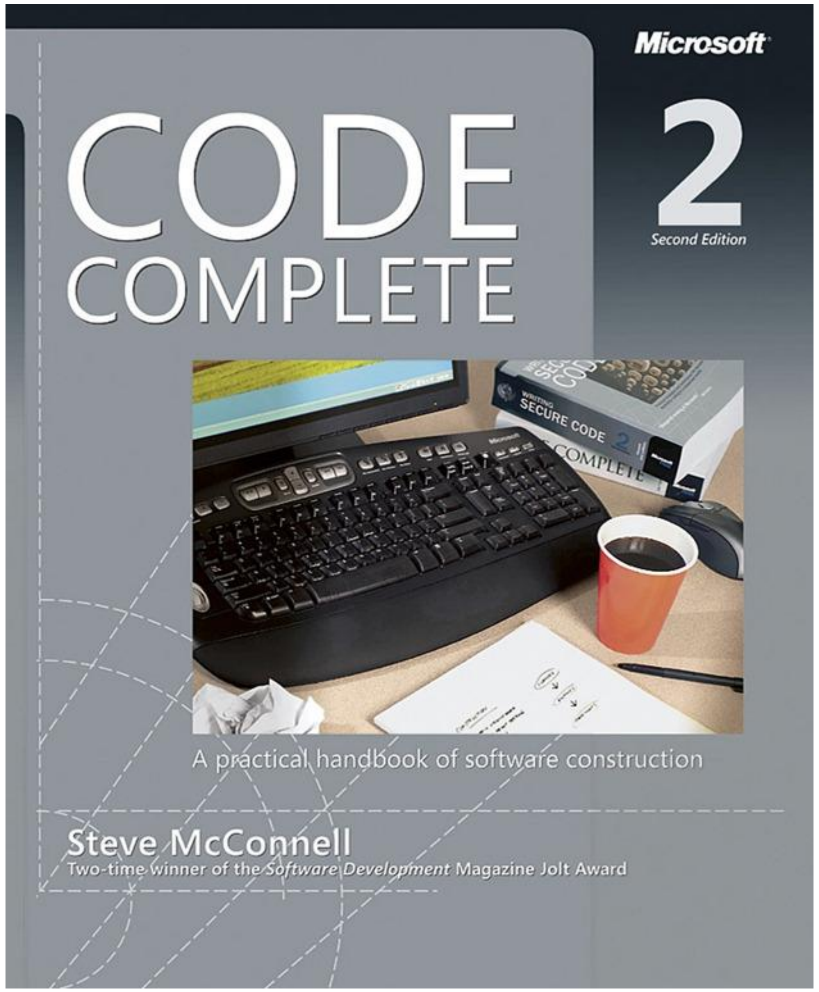
A Handbook of Agile Software Craftsmanship

Foreword by James O. Coplien

Robert C. Martin

ALWAYS LEARNING

PEARSON


Microsoft

CODE COMPLETE

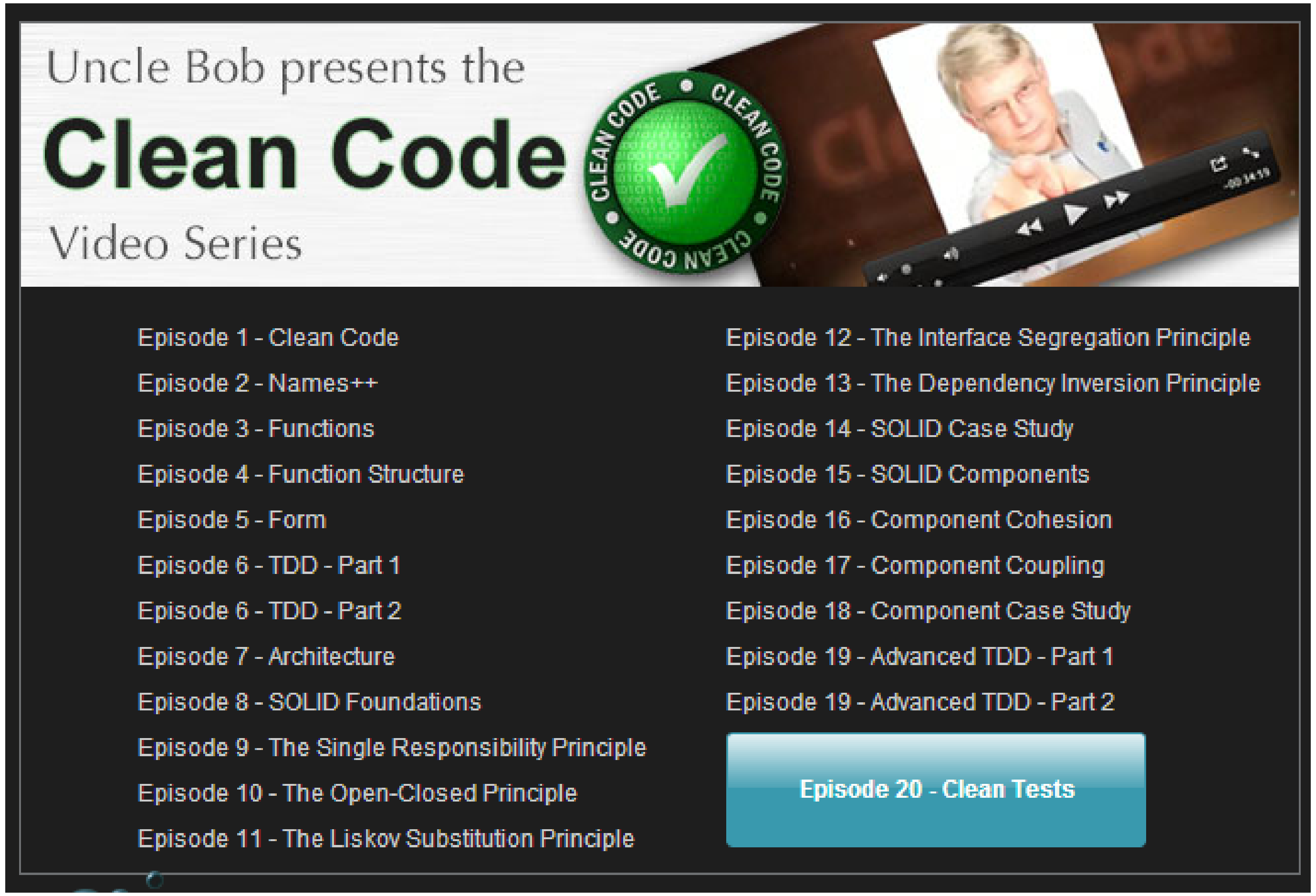
2 Second Edition

A practical handbook of software construction

Steve McConnell

Two-time winner of the *Software Development* Magazine Jolt Award

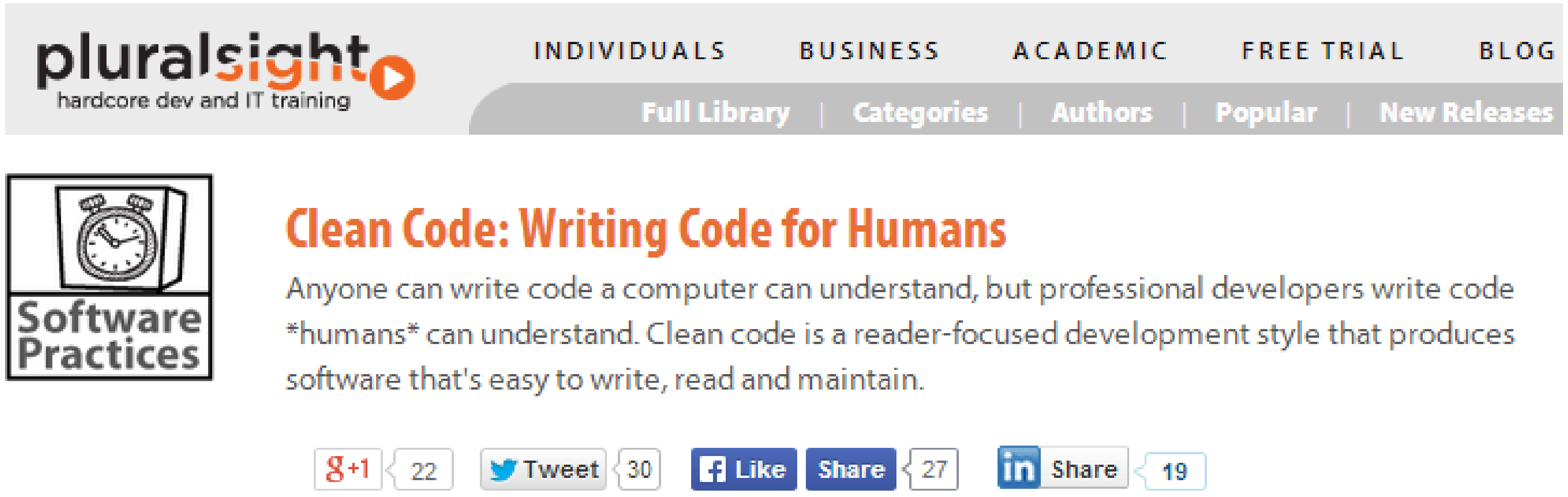
Where to Go Next



Uncle Bob presents the
Clean Code
Video Series

Episode 1 - Clean Code
Episode 2 - Names++
Episode 3 - Functions
Episode 4 - Function Structure
Episode 5 - Form
Episode 6 - TDD - Part 1
Episode 6 - TDD - Part 2
Episode 7 - Architecture
Episode 8 - SOLID Foundations
Episode 9 - The Single Responsibility Principle
Episode 10 - The Open-Closed Principle
Episode 11 - The Liskov Substitution Principle
Episode 12 - The Interface Segregation Principle
Episode 13 - The Dependency Inversion Principle
Episode 14 - SOLID Case Study
Episode 15 - SOLID Components
Episode 16 - Component Cohesion
Episode 17 - Component Coupling
Episode 18 - Component Case Study
Episode 19 - Advanced TDD - Part 1
Episode 19 - Advanced TDD - Part 2
Episode 20 - Clean Tests

Where to Go Next



Clean Code: Writing Code for Humans

Anyone can write code a computer can understand, but professional developers write code *humans* can understand. Clean code is a reader-focused development style that produces software that's easy to write, read and maintain.

http://pluralsight.com/training/Courses/TableOfContents/writing-clean-code-humans

Where to Go Next

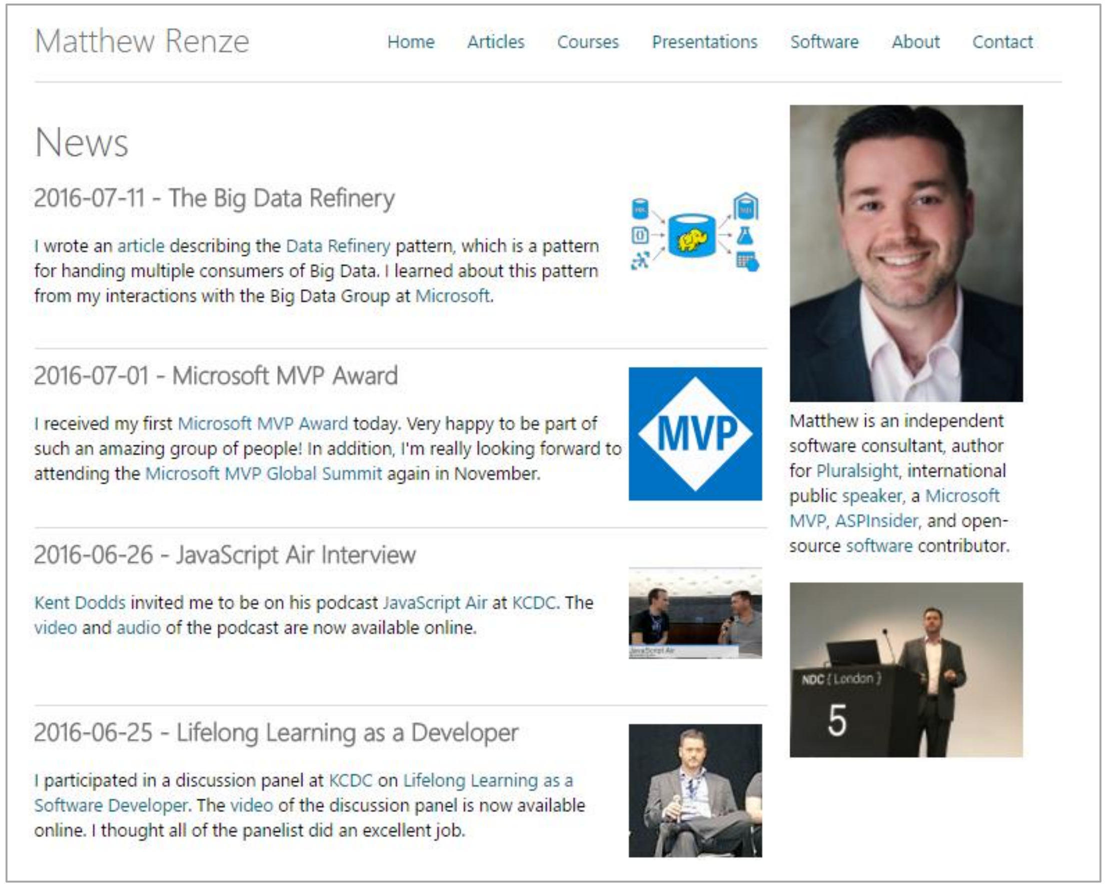
Articles

Courses

Presentations

Source Code

Videos



Matthew Renze

News

2016-07-11 - The Big Data Refinery

I wrote an article describing the Data Refinery pattern, which is a pattern for handing multiple consumers of Big Data. I learned about this pattern from my interactions with the Big Data Group at Microsoft.

2016-07-01 - Microsoft MVP Award

I received my first Microsoft MVP Award today. Very happy to be part of such an amazing group of people! In addition, I'm really looking forward to attending the Microsoft MVP Global Summit again in November.

2016-06-26 - JavaScript Air Interview

Kent Dodds invited me to be on his podcast JavaScript Air at KCDC. The video and audio of the podcast are now available online.

2016-06-25 - Lifelong Learning as a Developer

I participated in a discussion panel at KCDC on Lifelong Learning as a Software Developer. The video of the discussion panel is now available online. I thought all of the panelist did an excellent job.

Matthew is an independent software consultant, author for Pluralsight, international public speaker, a Microsoft MVP, ASPInsider, and open-source software contributor.

www.matthewrenze.com

Feedback

Feedback is very important to me!

One thing you liked?

One thing I could improve?

“Programming is not about telling the computer what to do.

Programming is the art of telling another human what the computer should do.”

- Donald Knuth

“Any fool can write code that a computer can understand.

Good programmers write code that humans can understand.”

- Martin Fowler

Uncle Bob Wants You:

“To leave the campground just a little bit cleaner than you found it.”

Contact Info

Matthew Renze

Data Science Consultant

Renze Consulting

Twitter: @matthewrenze

Email: info@matthewrenze.com

Website: www.matthewrenze.com

Thank You! :)