# Clean Code

## Homicidal Maniacs Read Code, Too

Presented by Jeremy Clark
www.jeremybytes.com

# What is Clean Code?

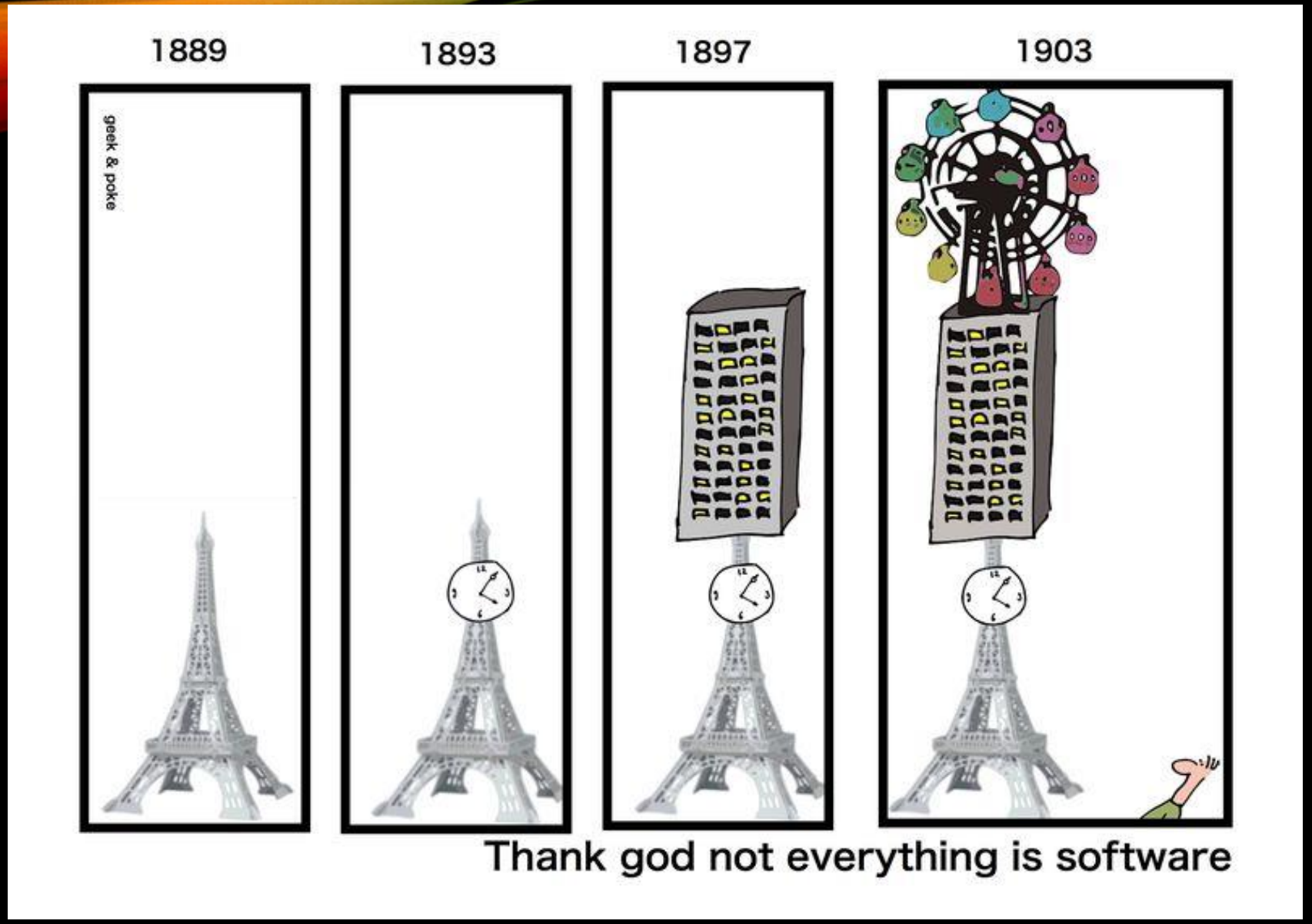
- Readable
- Maintainable
- Testable
- Elegant

©Jeremy Clark 2015

# Why Do We Care?

There's no such thing as write-once code

©Jeremy Clark 2015




http://geekandpoke.typepad.com/geekandpoke/2012/03/thank-god-not-everything-is-software.html

# Why Do We Care?

There's no such thing as write-once code

- Bug Fixes
- Business Changes
- Enhancements
- New Functionality

©Jeremy Clark 2015

# What Prevents Clean Code?

- Ignorance
- Stubbornness
- Short-Timer Syndrome
- Arrogance
- Job Security
- Scheduling

©Jeremy Clark 2015

# What Prevents Clean Code?

Number one reason:

"I'll clean it up later."

Pro Tip: "Later" never comes.

©Jeremy Clark 2015

# The Truth about Clean Code

- Clean Code saves time.
- We can't take a short-term view of software.
- We need to look at the lifespan of the application.

©Jeremy Clark 2015

# How Do You Write Clean Code?

- Rule of Thumb:

Imagine that the developer
who comes after you
is a homicidal maniac
who knows where you live.

-Unknown

©Jeremy Clark 2015

# The Next Developer

©Jeremy Clark 2015

# This Might Take Awhile

©Jeremy Clark 2015

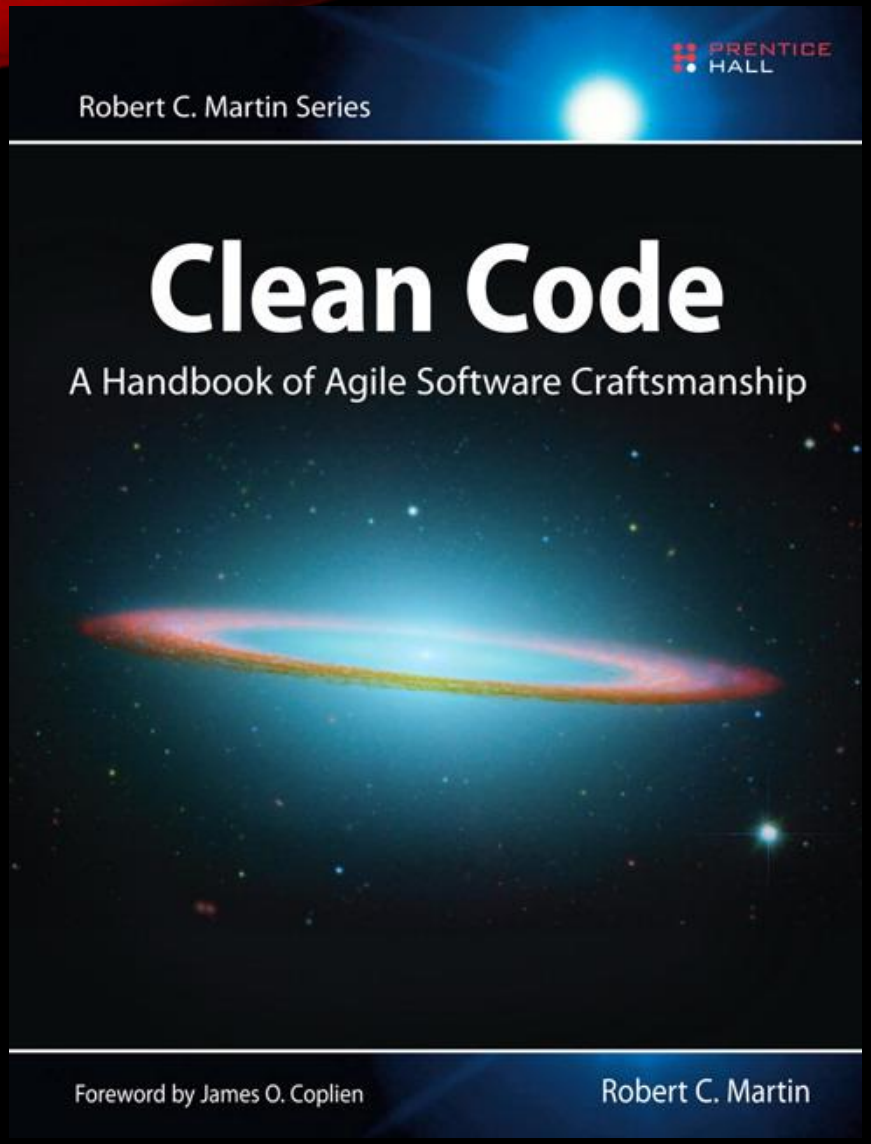
# The Problem

- Readable (by mere mortals)
- Maintainable
- Testable
- Elegant

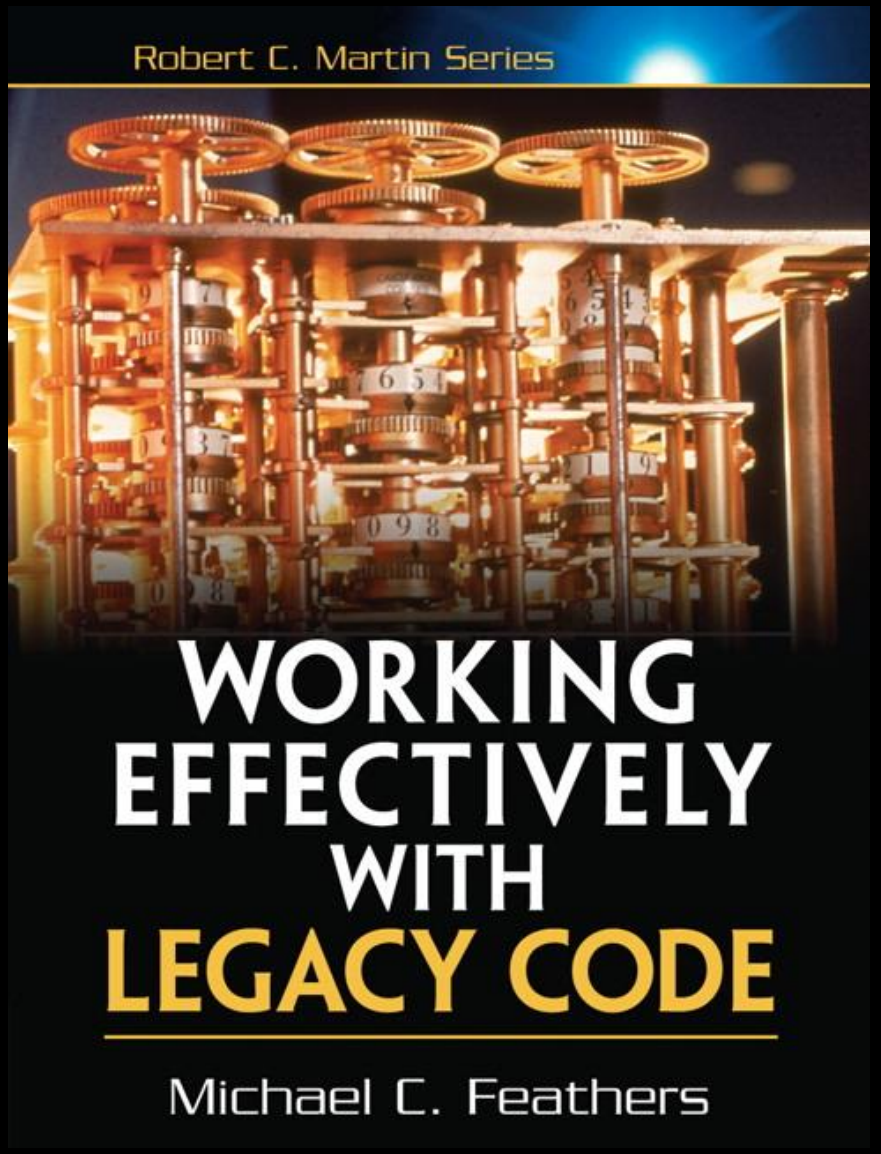
All of these qualities are subjective.

©Jeremy Clark 2015

Robert C. Martin




Michael C. Feathers




©Jeremy Clark 2015

## The Dry Principle

# Don't Repeat Yourself

- copy/pasta = spaghetti code

©Jeremy Clark 2015

# Naming

©Jeremy Clark 2015

# Intentional Naming

- `theList`
  - Not very good
- `ProductList`
  - A bit better
- `ProductCatalog`
  - Good

©Jeremy Clark 2015

# Naming

- Use Nouns for Variables, Properties, Parameters
  - `indexer`, `currentUser`, `PriceFilter`
- Use Verbs for Methods and Functions
  - `SaveOrder()`, `getDiscounts()`, `RunPayroll()`
- Pronounceable and Unambiguous
  - `recdptrl` = received patrol? record department role?

©Jeremy Clark 2015

# Naming Standard

- Camel Case?
- Pascal Case?
- Lower Case with Underscores?

It doesn't matter

# Have a Standard
# Be Consistent

©Jeremy Clark 2015

# Comments

```
// Determine if End of Day Time for Last Date
// has been reached
// If Last Date is null use Converted Date
// Based on Today's Date > Last Date
//  And Curr Time >= End of Day Time
```

©Jeremy Clark 2015

# Comments

- Rule #1: Comments lie
  - Code is updated or moved, but not the comments

©Jeremy Clark 2015

# Comments Lie

©Jeremy Clark 2015

# Comments

- Rule #1: Comments lie
  - Code is updated or moved, but not the comments
- Rule #2: Comments do not make up for bad code
  - If the code is that unclear, rewrite the code

©Jeremy Clark 2015

# Good Comments

- Can be used to describe intent or clarification
  - Ex: // Sample input: Oct 5, 2015 - 13:54:15 PDT
- Can be used to give warnings or consequences
  - Ex:
```
// We do a deep copy of this collection to make
// sure that updates to one copy do not affect
// the other
```

©Jeremy Clark 2015

# Good Comments

- Can be used for TODOs
  - Especially useful when the IDE supports it
  - These should be temporary

©Jeremy Clark 2015

# Bad Comments

- Avoid “journaling” comments
  - Ex: // 03/20/1996 - jjc - Added tax calculation
  - This is what source control is for: Who, What, When

©Jeremy Clark 2015

# Know Your Tools

©Jeremy Clark 2015

# Bad Comments

- Avoid “journaling” comments
  - Ex: // 03/20/1996 - jjc - Added tax calculation
  - This is what source control is for: Who, What, When
- Avoid “noise” comments
  - Ex: // Default constructor

©Jeremy Clark 2015

# Bad Comments

- Do not comment out code
  - Code no longer in use should be deleted
  - If needed, you can always retrieve it from source control

©Jeremy Clark 2015

# Know Your Tools

©Jeremy Clark 2015

# Functions and Methods

©Jeremy Clark 2015

# Functions and Methods

- Keep methods short
  - Should fit on a single screen
  - Prefer methods no longer than 10 lines

# Do one thing!

©Jeremy Clark 2015

# Multiple Levels of Methods

- High level
  - Overview of functionality
- Mid-level
  - More details, but not too deep
- Detail
  - The “weeds” of the functionality

©Jeremy Clark 2015

# Work in Small Chunks

If you aren't writing incremental code, you are writing excremental code.

©Jeremy Clark 2015

# What is Refactoring?

Making code better
without changing the functionality

©Jeremy Clark 2015

# Refactoring and Unit Testing

- If you don't have unit tests,
  you don't know what your code does.

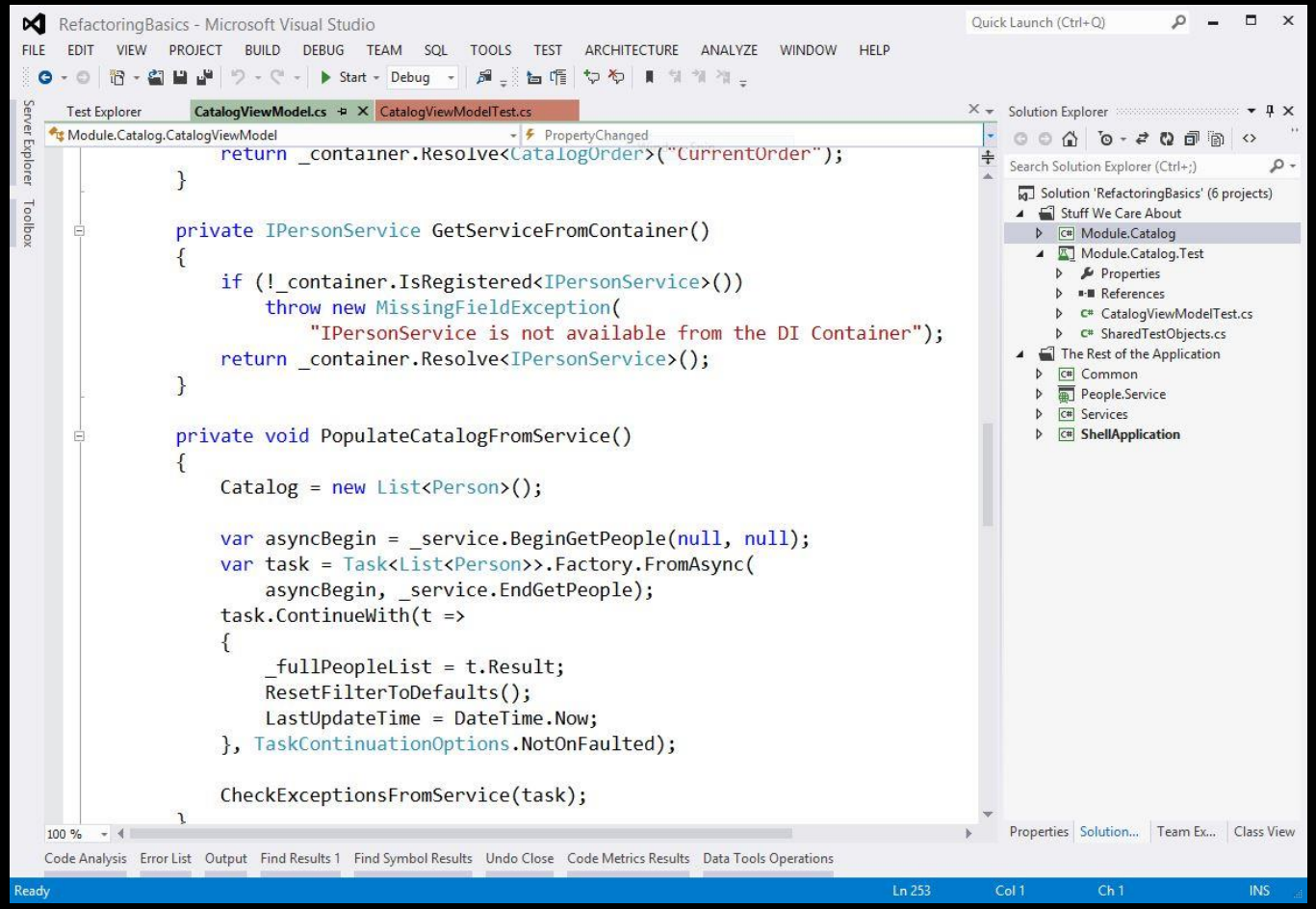
- Refactoring Step 1:
  - Bring your code under test.
- Refactoring Step 2:
  - Safely and confidently update the code.

©Jeremy Clark 2015

# The Watcher

©Jeremy Clark 2015

# Making Code Cleaner




©Jeremy Clark 2015

# Be a Clean Code Advocate

The Boy Scout Rule

Always leave the campground cleaner than you found it.

The Clean Coder Rule

Always leave the code cleaner than you found it.

©Jeremy Clark 2015

# Thank You!

## Jeremy Clark

- http://www.jeremybytes.com
- jeremy@jeremybytes.com
- @jeremybytes

©Jeremy Clark 2015