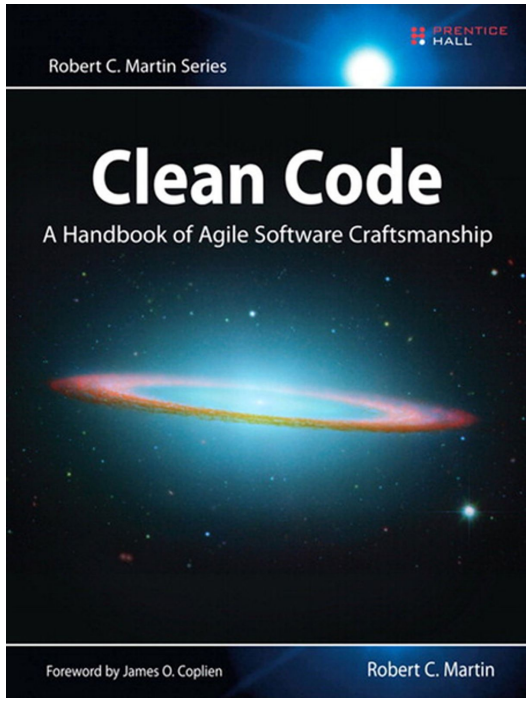
CLEAN CODE

Presented By Amirhossein Sadr
AP SPRING 2024 Course Lectured By Dr. Mojtaba Vahidi Asl

Two Reasons

1. You are a *programmer*

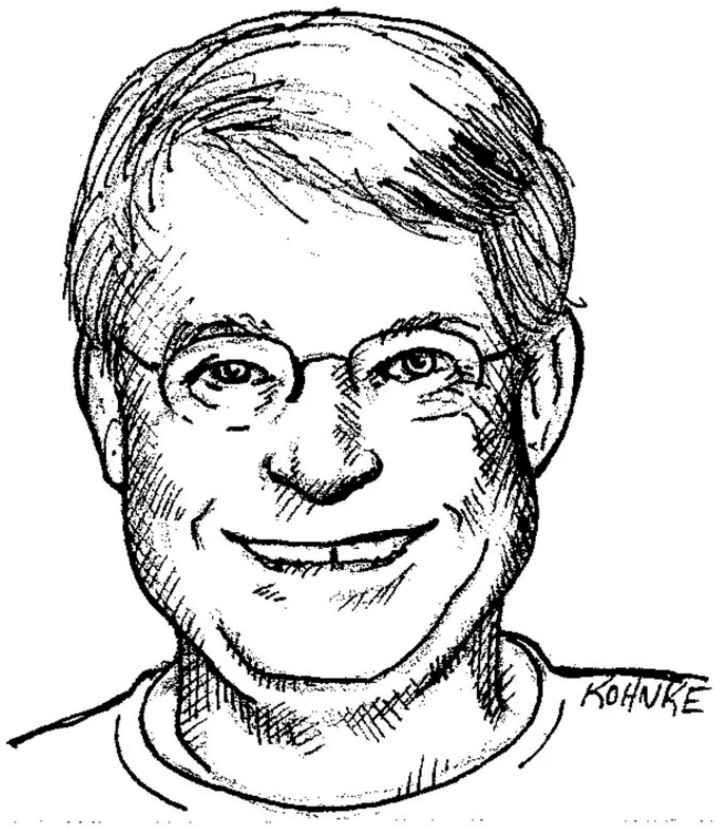
2. You want to be a *better programmer*



The Boy Scout Rule

Robert C. Martin

Uncle Bob



Elegance

I like my code to be elegant and efficient

Clean code does one thing well

Bjarne Stroustrup

Simple, direct, prose

Clean code is simple and direct

Clean code reads like well-written prose

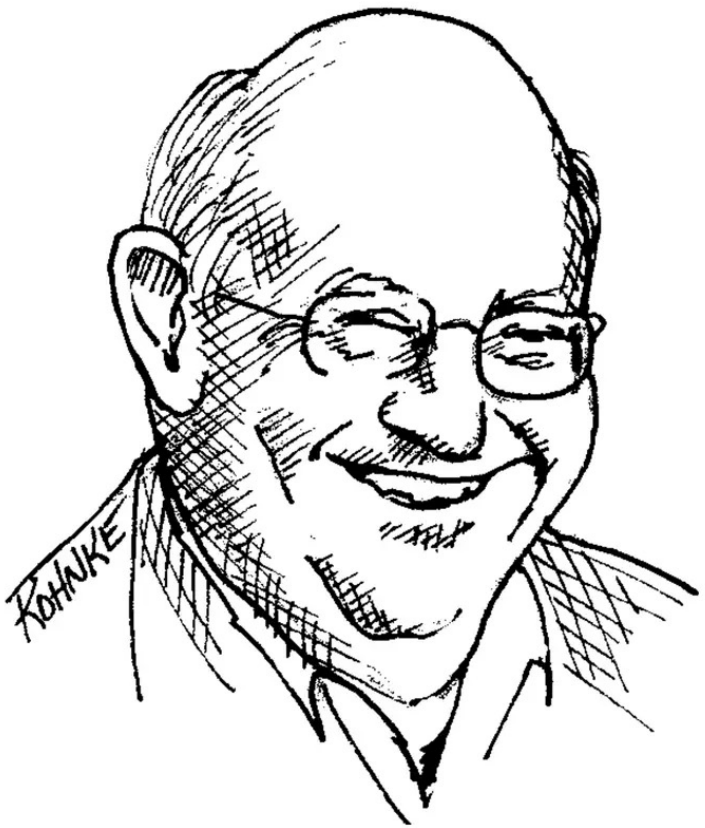
Grady Booch

Literate

Clean code can be read

Clean code should be literate

Dave Thomas



Care

Clean code always
looks like it was written
by someone who cares

Michael Feathers

Small, expressive, simple

Reduced Duplication,
High Expressiveness
and early building of
simple abstractions

Ron Jeffries

What you expected

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

Ward Cunnigham

Meaningful Names

Use Intention-Revealing Names

```
public List<int[]> getThem() {
    List<int[]> list1 = new ArrayList<int[]>();
    for (int[] x : theList)
        if (x[0] == 4)
            list1.add(x);
    return list1;
}
```

Meaningful Names

Use Intention-Revealing Names

```
public List<Cell> getFlaggedCells() {
    List<Cell> flaggedCells = new ArrayList<Cell>();
    for (Cell cell : gameBoard)
        if (cell.isFlagged())
            flaggedCells.add(cell);
    return flaggedCells;
}
```

Meaningful Names

Avoid Disinformation

```
int a = l;
if (O == l)
    a = O1;
else
    l = O1;
```

Make Meaningful Distinctions

```
public static void copyChars(char a1[], char a2[]) {
    for (int i = 0; i < a1.length; i++) {
        a2[i] = a1[i];
    }
}
```

Meaningful Names

Use Pronounceable Names

```
class DtaRcrd102 {
    private Date genymdhms;
    private Date modymdhms;
    private final String pszqint = "102";
    /* ... */
};
```

Meaningful Names

Use Pronounceable Names

```
class Customer {
    private Date generationTimestamp;
    private Date modificationTimestamp;;
    private final String recordId = "102";
    /* ... */
};
```

Meaningful Names

Use Searchable Names

```
for (int j = 0; j < 34; j++) {
    s += (t[j] * 4) / 5;
}
```

Meaningful Names

Use Searchable Names

```
int realDaysPerIdealDay = 4;
const int WORK_DAYS_PER_WEEK = 5;
int sum = 0;
for (int j = 0; j < NUMBER_OF_TASKS; j++) {
    int realTaskDays = taskEstimate[j] *
realDaysPerIdealDay;
    int realTaskWeeks = (realdays / WORK_DAYS_PER_WEEK);
    sum += realTaskWeeks;
}
```

Meaningful Names

Member Prefixes (Avoid encodings)

```
public class Part {
    private String m_dsc; // The textual description
    void setName(String name) {
        m_dsc = name;
    }
}
```

Hungarian Notation (Avoid encodings)

```
PhoneNumber phoneString;
// name not changed when type changed!
```

Meaningful Names

Member Prefixes (Avoid encodings)

```
public class Part {
    String description;
    void setDescription(String description) {
        this.description = description;
    }
}
```

Hungarian Notation (Avoid encodings)

```
PhoneNumber phone;
```

Meaningful Names

Avoid Mental Mapping

```
for (a = 0; a < 10; a++)
    for (b = 0; b < 10; b++)
```

Class Names

```
// a class name should not be a verb
```

Meaningful Names

Avoid Mental Mapping

```
for (i = 0; i < 10; i++)
    for (j = 0; j < 10; j++)
```

Class Names

```
Customer, WikiPage, Account, AddressParser
// a class name should not be a verb
```

Meaningful Names

Method Names

```
postPayment, deletePage, save
// methods should have verb or verb phrase names


string name = employee.getName();
customer.setName("mike");
if (paycheck.isPosted())...


Complex fulcrumPoint = Complex.fromRealNumber(23.0);
// is generally better than
Complex fulcrumPoint = new Complex(23.0);
```

Meaningful Names

Pick One Word Per Concept

```
fetch, retrieve, get    // as equivalent methods

controller, manager, driver  // confusing
```

Don't Pun

```
// avoid using the same word for two purposes
```

Meaningful Names

Use Solution Domain Names

```
AccountVisitor, JobQueue
// people who read your code will be programmers
```

Add Meaningful Context

```
firstName, lastName, street, city, state, zipcode
// a better solution
addrFirstName, addrLastName, addrState
// a better solution
Class Address
```

Meaningful Names

Don't Add Gratuitous Context

```
Address
// is a fine name for a class

AccountAddress, CustomerAddress
// are fine names for instances of the class Address
// but could be poor names for classes


MAC addresses, port addresses, Web addresses
// a better solution
PostalAddress, MAC, URI
```

Functions

Small!

```
// rules of functions:
//     1. should be small
//     2. should be smaller than that

// < 150 characters per line
// < 20 lines
```

Do One Thing

```
// FUNCTIONS SHOULD DO ONE THING. THEY SHOULD DO IT WELL.
// THEY SHOULD DO IT ONLY.
```

Functions

One Level of Abstraction per Function

```
// high level of abstraction
   getHtml()

// intermediate level of abstraction
   String pagePathName = PathParser.render(pagePath);

// remarkably low level
   .append("\n")
```

Reading Code from Top to Bottom

```
// the Stepdown Rule
```

Functions

Switch Statements

```
class Employee...
    int payAmount() {
        switch (getType()) {
            case EmployeeType.ENGINEER:
                return _monthlySalary;
            case EmployeeType.SALESMAN:
                return _monthlySalary + _commission;
            case EmployeeType.MANAGER:
                return _monthlySalary + _bonus;
            default:
                throw new Exception("Incorrect Employee");
        }
    }
```

Functions

Switch Statements

```
class EmployeeType...
    abstract int payAmount(Employee emp);

class Salesman...
    int payAmount(Employee emp) {
        return emp.getMonthlySalary() + emp.getCommission();
    }

class Manager...
    int payAmount(Employee emp) {
        return emp.getMonthlySalary() + emp.getBonus();
    }
```

Functions

Use Descriptive Names

```
testableHtml  =>  includeSetupAndTeardownPages

includeSetupAndTeardownPages, includeSetupPages,
includeSuiteSetupPage, includeSetupPage
// what happened to
includeTeardownPages, includeSuiteTeardownPage,
includeTeardownPage
```

Function Arguments

```
// the ideal number of arguments for a function is zero
```

Functions

Common Monadic Forms

```
// if a function is going to transform its input argument,
// the transformation should appear as the return value

StringBuffer transform(StringBuffer in)
// is better than
void transform(StringBuffer out)
```

Flag Arguments

```
render(true)
```

Functions

Common Monadic Forms

```
// asking a question about that argument
boolean fileExists("MyFile")

// operating on that argument, transforming and returning it
InputStream fileOpen("MyFile")

// event, use the argument to alter the state of the system
void passwordAttemptFailedNtimes(int attempts)
```

Flag Arguments

```
renderForSuite()
renderForSingleTest()
```

Functions

Dyadic Functions

```
writeField(name)
// is easier to understand than
writeField(outputStream, name)

// perfectly reasonable
Point p = new Point(0,0)

// problematic
assertEquals(expected, actual)
```

Triads

```
assertEquals(message, expected, actual)
```

Functions

Argument Objects

```
Circle makeCircle(double x, double y, double radius);
Circle makeCircle(Point center, double radius);
```

Verbs and Keywords

```
write(name)
writeField(name)

assertEquals(expected, actual)
assertExpectedEqualsActual(expected, actual)
```

Functions

Have No Side Effects

```
// do something or answer something, but not both
public boolean set(String attribute, String value);

setAndCheckIfExists

if (attributeExists("username")) {
    setAttribute("username", "unclebob");
    ...
}
```

Functions

Don't Repeat Yourself (DRY)

```
// duplication may be the root of all evil in software
```

Structured Programming

```
// Edsger Dijkstra's rules
//   one entry
//   one exit

// functions small
// occasional multiple return, break, or continue statement
// can sometimes even be more expressive Dijkstra's rules
```

Comments

Comments Do Not Make Up for Bad Code

```
// don't comment bad code, rewrite it!
```

Explain Yourself in Code

```
// Check to see if the employee is eligible for full
benefits
if ((employee.flags & HOURLY_FLAG) &&
   (employee.age > 65))

if (employee.isEligibleForFullBenefits())
```

Comments (good)

Legal Comments

```
// Copyright (C) 2011 by Osoco. All rights reserved.
// Released under the terms of the GNU General Public
License // version 2 or later.
```

Informative Comments

```
// Returns an instance of the Responder being tested.
protected abstract Responder responderInstance();
// renaming the function: responderBeingTested

// format matched kk:mm:ss EEE, MMM dd, yyyy
Pattern timeMatcher = Pattern.compile(
"\\d*:\\d*:\\d* \\w*, \\w* \\d*, \\d*");
```

Comments (good)

Explanation of Intent

```
//This is our best attempt to get a race condition
//by creating large number of threads.
for (int i = 0; i < 25000; i++) {
    WidgetBuilderThread widgetBuilderThread =
    new WidgetBuilderThread(widgetBuilder, text, failFlag);
    Thread thread = new Thread(widgetBuilderThread);
    thread.start();
}
```

Clarification

```
assertTrue(a.compareTo(b) == -1); // a < b
assertTrue(b.compareTo(a) == 1); // b > a
```

Comments (good)

Warning of Consequences

```
public static SimpleDateFormat makeStandardHttpDateFormat() {
    //SimpleDateFormat is not thread safe,
    //so we need to create each instance independently.
    SimpleDateFormat df = new SimpleDateFormat("dd MM yyyy");
    df.setTimeZone(TimeZone.getTimeZone("GMT"));
    return df;
}
```

TODO Comments

```
//TODO-MdM these are not needed
// We expect this to go away when we do the checkout model
```

Comments (good)

Amplification

```
String listItemContent = match.group(3).trim();
// the trim is real important. It removes the starting
// spaces that could cause the item to be recognized
// as another list.
new ListItemWidget(this, listItemContent, this.level + 1);
return buildList(text.substring(match.end()));
```

Comments (bad)

Mumbling

```
try {
    String propertiesPath = propertiesLocation + "/" +
                            PROPERTIES_FILE;
    FileInputStream propertiesStream =
        new FileInputStream(propertiesPath);
    loadedProperties.load(propertiesStream);
}
catch(IOException e) {
    // No properties files means all defaults are loaded
}
```

Comments (bad)

Redundant Comments

```
// Utility method that returns when this.closed is true.
// Throws an exception if the timeout is reached.
public synchronized void waitForClose
                (final long timeoutMillis) throws Exception
{
    if(!closed) {
        wait(timeoutMillis);
        if(!closed)
            throw new Exception("MockResponseSender
                                 could not be closed");
    }
}
```

Comments (bad)

Redundant Comments

```
/**
 * The processor delay for this component.
 */
protected int backgroundProcessorDelay = -1;

/**
 * The lifecycle event support for this component.
 */
protected LifecycleSupport lifecycle =
                                    new LifecycleSupport(this);
/**
 * The container event listeners for this Container.
 */
protected ArrayList listeners = new ArrayList();
```

Comments (bad)

Mandated Comments

```
/**
 * @param title The title of the CD
 * @param author The author of the CD
 * @param tracks The number of tracks on the CD
 * @param durationInMinutes The duration of the CD in minutes
 */
public void addCD(String title, String author,
                  int tracks, int durationInMinutes) {
    CD cd = new CD();
    cd.title = title;
    cd.author = author;
    cd.tracks = tracks;
    cd.duration = durationInMinutes;
}
```

Comments (bad)

Journal Comments

```
* Changes (from 11-Oct-2001)
* --------------------------
* 11-Oct-2001 : Re-organised the class and moved it to new
*               package com.jrefinery.date (DG);
* 05-Nov-2001 : Added a getDescription() method, and
*               eliminated NotableDate class (DG);
* 12-Nov-2001 : IBD requires setDescription() method, now
*               that NotableDate class is gone (DG); Changed
*               getPreviousDayOfWeek(),
*               getFollowingDayOfWeek() and
*               getNearestDayOfWeek() to correct bugs (DG);
* 05-Dec-2001 : Fixed bug in SpreadsheetDate class (DG);
* 29-May-2002 : Moved the month constants into a separate
*               interface (MonthConstants) (DG);
```

Comments (bad)

Noise Comments

```
/**
 * Default constructor.
 */
protected AnnualDateRule() { }

/** The day of the month. */
private int dayOfMonth;

/**
 * Returns the day of the month.
 * @return the day of the month.
 */
public int getDayOfMonth() {
    return dayOfMonth;
}
```

Comments (bad)

Scary Noise

```
/** The name. */
private String name;

/** The version. */
private String version;

/** The licenceName. */
private String licenceName;

/** The version. */
private String info;
```

Comments (bad)

Position Markers

```
// Actions ////////////////////////////////////
```

Closing Brace Comments

```
while ((line = in.readLine()) != null) {
    lineCount++;
    charCount += line.length();
    String words[] = line.split("\\W");
    wordCount += words.length;
} //while
```

Comments (bad)

Attributions and Bylines

```
/* Added by Rick */
```

Commented Out Code

```
InputStreamResponse response = new InputStreamResponse();
response.setBody(formatter.getResultStream(),
formatter.getByteCount());
// InputStream resultsStream = formatter.getResultStream();
// StreamReader reader = new StreamReader(resultsStream);
// response.setContent(reader.read(formatter.getByteCount()));
```

Comments (bad)

Nonlocal Information

```
/**
 * Port on which fitnesse would run. Defaults to <b>8082</b>.
 *
 * @param fitnessePort
 */
public void setFitnessePort(int fitnessePort)
{
    this.fitnessePort = fitnessePort;
}
```

Comments (bad)

Too Much Information

```
/*
  RFC 2045 - Multipurpose Internet Mail Extensions (MIME)
  Part One: Format of Internet Message Bodies section 6.8.
  Base64 Content-Transfer-Encoding
  The encoding process represents 24-bit groups of input bits
  as output strings of 4 encoded characters. Proceeding from
  left to right, a 24-bit input group is formed by
  concatenating 3 8-bit input groups.
  These 24 bits are then treated as 4 concatenated 6-bit
  groups, each of which is translated into a single digit in
  the base64 alphabet.
  When encoding a bit stream via the base64 encoding, the bit
  stream must be presumed to be ordered with the most-
  significant-bit first.
 */
```

Comments (bad)

Inobvious Connection

```
/*
 * start with an array that is big enough to hold all the
 * pixels (plus filter bytes), and an extra 200 bytes for
 * header info
 */
this.pngBytes = new byte[((this.width + 1) * this.height * 3)
                                    + 200];
```

Function Headers

```
// short functions don't need much description
```

Formatting

The Purpose of Formatting

```
// communication
```

The Newspaper Metaphor

```
// high-level -> details
```

Vertical Openness Between Concepts

```
// each blank line is a visual cue
// that identifies a new and separate concept
```

Formatting

Vertical Density

```
// vertical density implies close association

/**
 * The class name of the reporter listener
 */
private String m_className;

/**
 * The properties of the reporter listener
 */
private m_properties = new ArrayList();
```

Formatting

Vertical Distance

```
// variables
// should be declared as close to their usage as possible

// instance variables
// should be declared at the top of the class

// dependent functions
// if one function calls another, they should be vertically
// close, and the caller should be above the called
```

Formatting

Horizontal Openness and Density

```
private void measureLine(String line) {
    lineCount++;
    int lineSize = line.length();
    totalChars += lineSize;
    lineWidthHistogram.addLine(lineSize, lineCount);
    recordWidestLine(lineSize);
}

public static double root2(int a, int b, int c) {
    double determinant = determinant(a, b, c);
    return (-b - Math.sqrt(determinant)) / (2*a);
}
```

Formatting

Horizontal Alignment

```
public class FitNesseExpediter implements ResponseSender
{
    private   Socket          socket;
    private   InputStream     input;
    private   OutputStream    output;
    private   Request         request;
    private   Response        response;
    private   FitNesseContext context;
    protected long            requestParsingTimeLimit;
    private   long            requestProgress;
    private   long            requestParsingDeadline;
    private   boolean         hasError;
    ...
}
```

Formatting

Horizontal Alignment

```
public class FitNesseExpediter implements ResponseSender
{
    private Socket socket;
    private InputStream input;
    private OutputStream output;
    private Request request;
    private Response response;
    private FitNesseContext context;
    protected long requestParsingTimeLimit;
    private long requestProgress;
    private long requestParsingDeadline;
    private boolean hasError;
    ...
}
```

Formatting

Breaking Indentation

```
public class CommentWidget extends TextWidget {
    public static final String REGEXP =
            "^#[^\r\n]*(?:(?:\r\n)|\n|\r)?";
    public CommentWidget(String text) { super(text); }
    public String render() throws Exception { return ""; }
}
```

Formatting

Breaking Indentation

```
public class CommentWidget extends TextWidget {
    public static final String REGEXP =
            "^#[^\r\n]*(?:(?:\r\n)|\n|\r)?";

    public CommentWidget(String text) {
        super(text);
    }

    public String render() throws Exception {
        return "";
    }
}
```

Formatting

Team Rules

```
// every programmer has his own favorite formatting rules
// but if he works in a team
// then the team rules
```

Classes

Class Organization

```
// public static constants
// private static variables
// private instance variables
// public functions
// private utilities called by a public function right after
```

Classes Should Be Small!

```
// the first rule is that they should be small
// the second rule is that they should be smaller than that
```

Classes

The Single Responsibility Principle (SRP)

```
// a class or module should have one, and only one,
// reason to change
```

Cohesion

```
// maintaining cohesion results in many small classes
```

Emergence

Simple Design Rule 1: Runs All the Tests

Simple Design Rules 2: No Duplication

Simple Design Rules 3: Expressive

Simple Design Rules 4: Minimal Classes and Methods

Questions?

Resources

Robert C. Martin, Clean Code (Prentice Hall, 2008)

Based on Arturo Herrero Slides

Thanks!