Software Development

Dr. Hamada I. AbdulWakel[1]

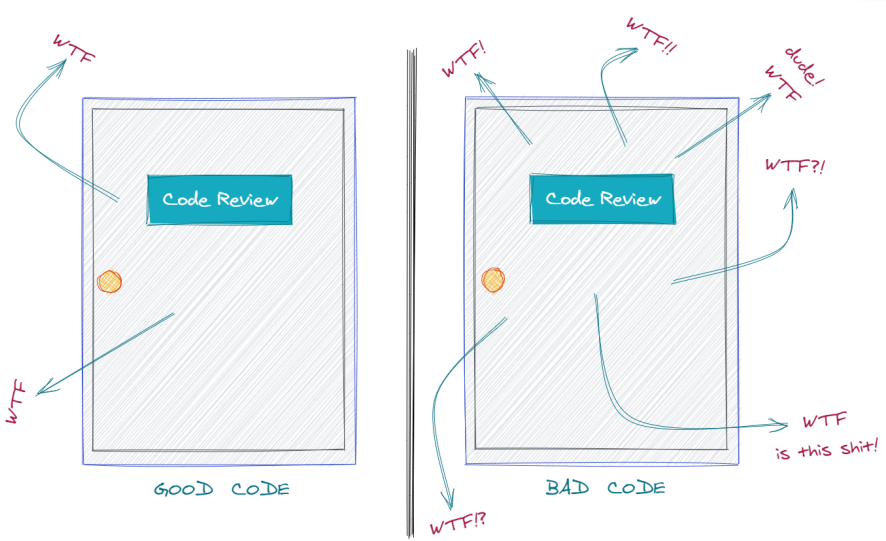
[1]Computer Science Department

2023 - 2022

Ch3: Code Construction Lec#4

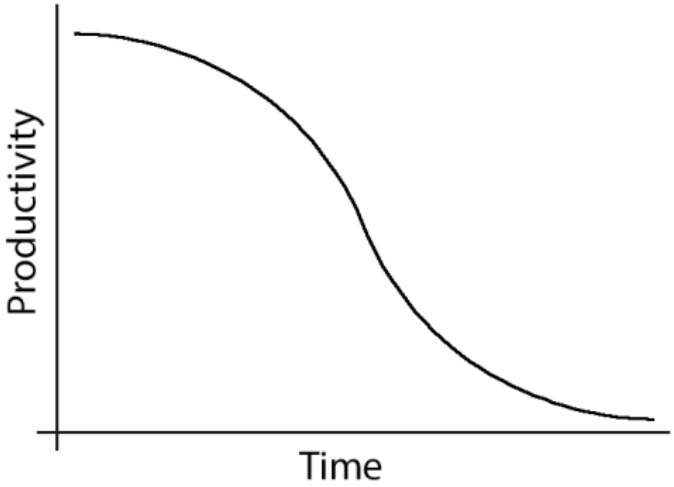
Uncle Bob??!!

Code Quality Measure



How Do You Write Clean Code?

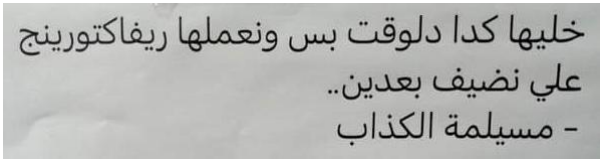
Cost of Bad Code



What is Clean Code?

Why Clean Code??

- Simplicity & clearness.
- Ability to respond to change requests.
- Easy to bring new resources into project.
- Logic behind the code.
- Time & Cost savings. **LeBlanc's law: Later equals never.**



- Easy to unit test code.
- Issue detection and resolution is more efficient.
- Reduce repeated code.

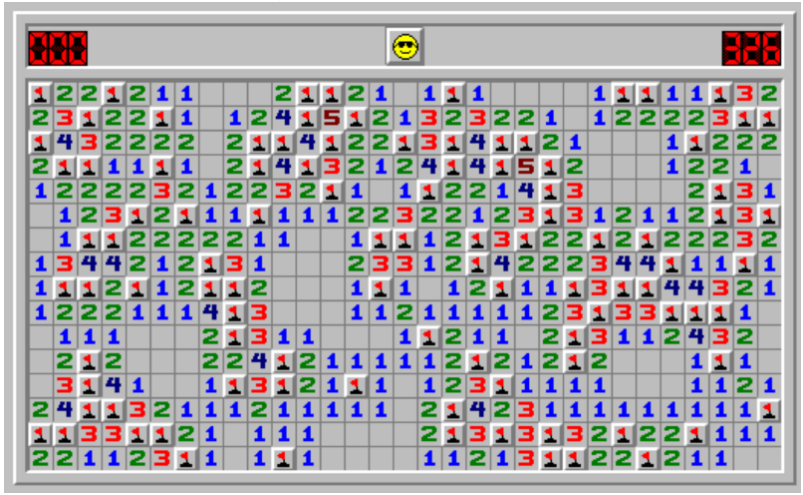
Clean Code, for Naming

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;
}
```

Clean Code, for Naming ...

A game of Minesweeper



Clean Code, for Naming ...

- Inject the "**domain language**" into your code.
  AccountVisitor, JobQuese
- Add meaningful context.
  firstName, lastName, street, city, state
  better solution is: class Address

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

Clean Code, for Naming ...

Avoid Disinformation

```
int a = l;
if (O == l)
    a = 01;
else
    l = 01;
```

Make Meaningful Distinctions

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

Clean Code, for Naming ...

- Don’t make the compiler happy. Don’t change names to make them work.

Use Pronounceable Names

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

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

Clean Code, for Naming ...

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

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;
}
```

Clean Code, for Naming ...

Member Prefixes (Avoid encodings)

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

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

Clean Code, for Naming ...

Hungarian Notation (Avoid encodings)

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

```
PhoneNumber phone;
```

Clean Code, for Naming ...

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
```

Method Names

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

Clean Code, for Naming ...

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
```

Clean Code, for 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.
```

Clean Code, for Functions ...

- Think of the "**What**" not the "**How**".

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
```

Clean Code, for Functions ...

Function Arguments

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

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)
```

Clean Code, for 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)
```

Clean Code, for 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)
```

Clean Code, for 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");
    ...
}
```

Clean Code, for 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");
        }
    }
```

Clean Code, for 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();
    }
```

Clean Code, for 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
```

```
int calculate(int x, int y){    return add(x, y);    }
int add(int x, int y){    return x + y;    }
```

Clean Code, for 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);
}
```

Clean Code, for Formatting ...

```
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;
    ...
}
```

Clean Code, for 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;
    ...
}
```

Clean Code, for Formatting ...

```
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 ""; }
}
```

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 "";
    }
}
```

Clean Code, for 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())
```

Clean Code, for 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.
```

// Delete production piece along with all its references.
public void delete(){ ..... }

Clean Code, for Comments (good) ...

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*");
```

Clean Code, for 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
```

Clean Code, for Comments (good) ...


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

Clean Code, for 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
}
```

Clean Code, for 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");
    }
}
```

Clean Code, for Comments (bad) ...

Redundant Comments

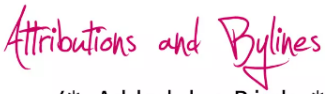
```
/**
 * The processor delay for this component.
 */
protected int backgroundProcessorDelay = -1;

/**
 * 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();
```

Clean Code, for 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;
}
```

Clean Code, for Comments (bad) ...




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


```
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()));
```

Clean Code, for Comments (bad) ...

Don't Use a Comment When You Can Use a Function or a Variable

```
// does the module from the global list <mod> depend on the
// subsystem we are part of?
if (smodule.getDependSubsystems()
    .contains(subSysMod.getSubSystem()))

// this could be rephrased without the comment as
ArrayList moduleDependees = smodule.getDependSubsystems();
String ourSubSystem = subSysMod.getSubSystem();
if (moduleDependees.contains(ourSubSystem))
```

Clean Code, for 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;
```

Clean Code, for 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);
```