STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

SSW 322: Software Engineering Design VI

Clean Code
2020 Spring

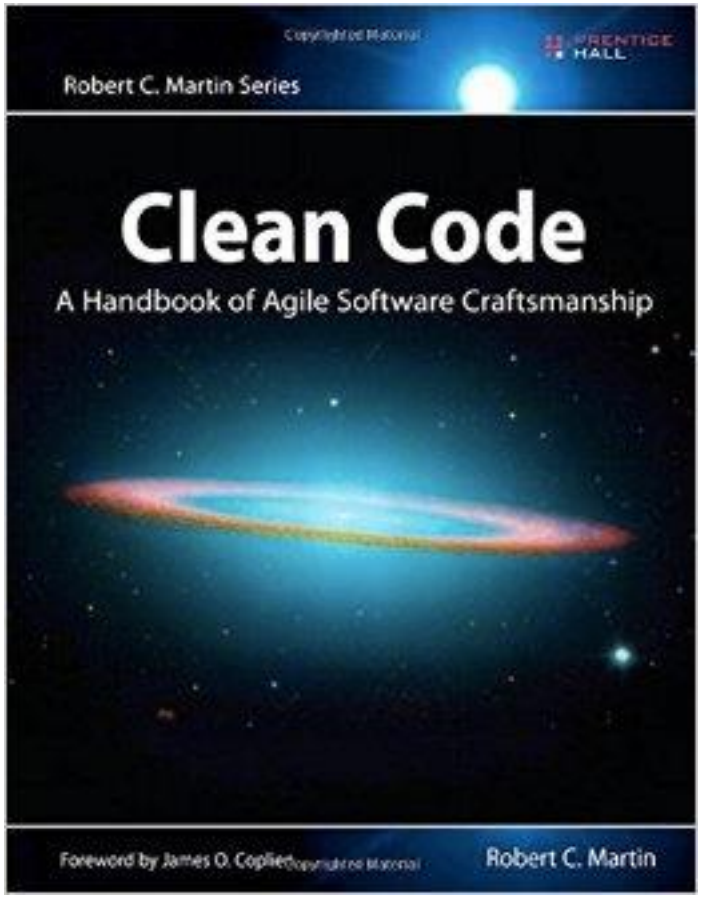
Prof. Lu Xiao
lxiao6@stevens.edu
Babbio 513
Office Hour: Monday/Wednesday 2 to 4 pm
Software Engineering
School of Systems and Enterprises

Today's Topics

- What is clean code?
- Why clean code?
- How to write clean code?
  - Use meaningful names
  - Avoid disinformation
  - Make meaningful distinctions
  - Use pronounceable names
  - Use searchable names
  - Use class/method naming conventions
  - Keep functions small
  - Use comments

Acknowledgement

- Materials in slides are from the following book:



Author:
Robert Cecil Martin (colloquially known as **Uncle Bob**) is an American software engineer and author. He is a co-author of the Agile Manifesto. He now runs a consulting firm called Uncle Bob Consulting LLC and Clean Coders which hosts videos based on his experiences and books.

What is clean code?

Bjarne Stroustrup
Inventor of C++

"I like my code to be elegant and efficient. The logic should be straightforward to make it hard for bugs to hide, the dependencies minimal to ease maintenance, error handling complete according to an articulated strategy, and performance close to optimal so as not to tempt people to make the code messy with unprincipled optimizations. Clean code does one thing well."

What is clean code?

Grady Booch

Author of Object Oriented Analysis and Design with Applications

"Clean code is simple and direct. Clean code reads like well-written prose. Clean code never obscures the designer's intent but rather is full of crisp [clearly defined] abstractions and straightforward lines of control."

What is clean code?

Dave Thomas

Founder of OTI, godfather of the Eclipse strategy

"Clean code can be read, and enhanced by a developer other than its original author. It has unit and acceptance tests. It has meaningful names. It provides one way rather than many ways for doing one thing. It has minimal dependencies, which are explicitly defined, and provides a clear and minimal API. Code should be literate since depending on the language, not all necessary information can be expressed clearly in code alone."

What is clean code?

Michael Feathers

Author of Working Effectively with Legacy Code

"I could list all of the qualities that I notice in clean code, but there is one overarching quality that leads to all of them. Clean code always looks it was written by someone who cares. There is nothing obvious that you can do to make it better. All of those things were thought about by the code's author, and if you try to imagine improvements, you're led back to where you are, sitting in appreciation of the code someone left for you – code left by someone who cares deeply about the craft."

What is clean code?

Ron Jeffries

Author of Extreme Programming Installed

"In recent years I begin, and nearly end, with Beck's rules of simple code. In priority order, simple code:

- Runs all tests
- Contains no duplication
- Expresses all the design ideas that are in the system
- Minimizes the number of entities such as classes, methods, functions, and the like."

What is clean code?

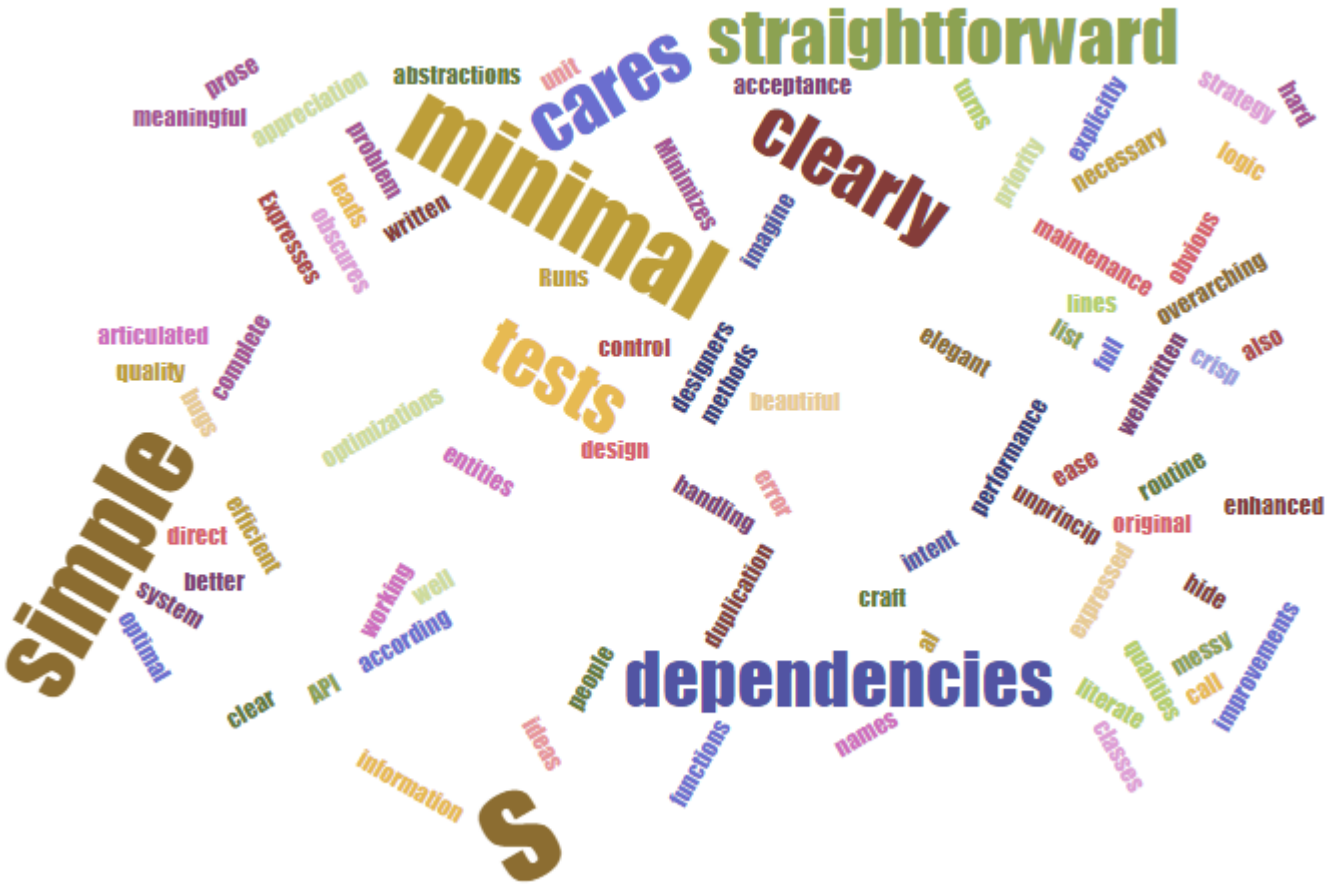
Ward Cunningham

Inventor of Wiki, Fit and much more

"Godfather of all those who care about code"

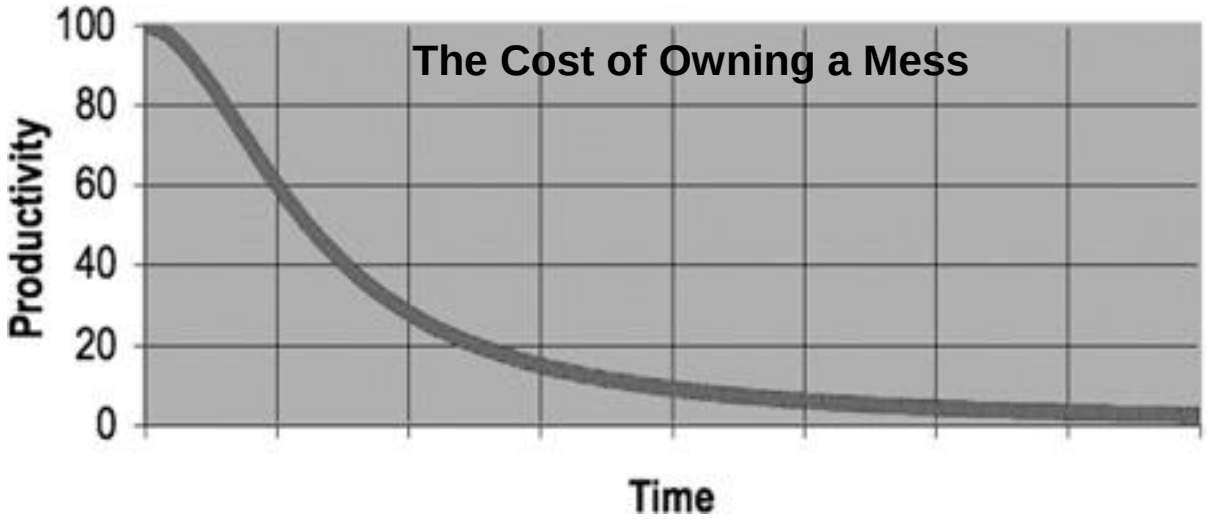
"You know you are working on clean code when each routine you read turns out to be pretty much what you expected. You can call it beautiful code when the codes also makes it look like the language was made for the problem."

What is clean code?



https://www.jasondavies.com/wordcloud/

Why clean code?



Teams that were moving very fast at the beginning of a project can find themselves moving at a snail's pace.

- Every change they make to the code breaks two or three other parts of the code. No change is trivial.
- Every addition or modification to the system requires that the tangles, twists, and knots be "understood" so that more tangles, twists, and knots can be added.
- Over time the mess becomes so big and so deep and so tall, they can not clean it up. There is no way at all.

Practices for clean code

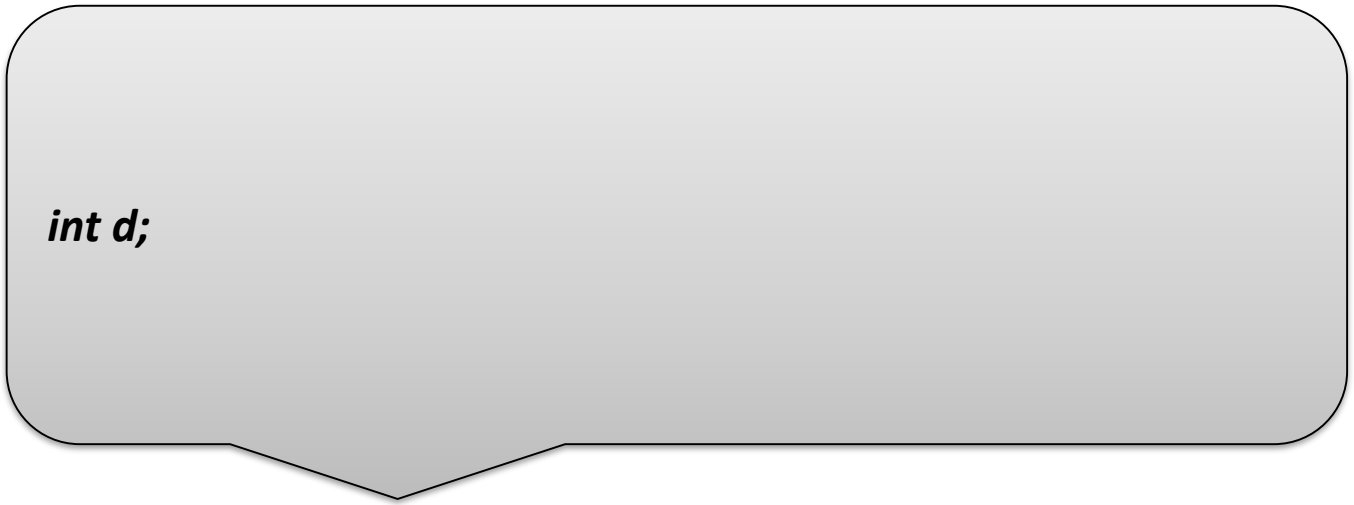
- Use meaningful names
- Avoid disinformation
- Make meaningful distinctions
- Use pronounceable names
- Use searchable names
- Use class/method naming conventions
- Keep functions small
- Use comments

Meaningful Names

- Names are everywhere in software.
- We name our **variables**, our **functions**, our **arguments**, **classes**, and **packages**.
- ***Names should reveal intent.***

Meaningful Names

- Choosing good names takes time but saves more than it takes.
- The name of a variable, function, or class, should answer all the big questions: why it exits, what it does, and how it is used?



Meaningful Names

- Choosing good names takes time but saves more than it takes.
- The name of a variable, function, or class, should answer all the big questions: why it exits, what it does, and how it is used?

```
int d;
// elapsed time in days
```

Meaningful Names

- Choosing good names takes time but saves more than it takes.
- The name of a variable, function, or class, should answer all the big questions: why it exits, what it does, and how it is used?

```
int d;
// elapsed time in days

int elapsedTimeInDays;
```

Avoid Disinformation

- Avoid leaving false clues that obscure the meaning of code.
- Beware of using names which vary in small ways.

```
accountList;
//when it is not a List
```

Avoid Disinformation

- Avoid leaving false clues that obscure the meaning of code.
- Beware of using names which vary in small ways.

~~*accountList;*~~
~~*//when it is not a List*~~

accountGroup;
bunchOfAccounts;

Avoid Disinformation

- Avoid leaving false clues that obscure the meaning of code.
- Beware of using names which vary in small ways.

XYZControllerForEfficientHandlingOfStrings
XYZControllerForEfficientStorageOfStrings

Avoid Disinformation

- Avoid leaving false clues that obscure the meaning of code.
- Beware of using names which vary in small ways.

How long does it take for you to spot the difference?

XYZControllerForEfficientHandlingOfStrings
XYZControllerForEfficientStorageOfStrings

Avoid Disinformation

- Avoid leaving false clues that obscure the meaning of code.
- Beware of using names which vary in small ways.

How long does it take for you to spot the difference?

XYZControllerForEfficientHandlingOfStrings
XYZControllerForEfficientStorageOfStrings

Avoid Disinformation

- Avoid leaving false clues that obscure the meaning of code.
- Beware of using names which vary in small ways.

A truly awful example of disinformation:

```
int a = l;
if (o == l)
a = 1o
else
l = 10;
```

Avoid Disinformation

- Avoid leaving false clues that obscure the meaning of code.
- Beware of using names which vary in small ways.

A truly awful example of disinformation:

```
int a = l;
if (o == l)
a = 1o
else
l = 10;
```

The use of lower-case L or o/O as variable names

Make Meaningful Distinctions

- Programmers create problems for themselves when they write code solely to satisfy a compiler or interpreter.

Make Meaningful Distinctions

- ***Number-series naming (a1,a2,…,an) is the opposite of intentional naming!!!!***
- Such names are not dis-informative—they are non informative; they provide no clue to the intention.

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

Make Meaningful Distinctions

- ***Number-series naming (a1,a2,…,an) is the opposite of intentional naming!!!!***
- Such names are not dis-informative—they are non informative; they provide no clue to the intention.

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

Make Meaningful Distinctions

- **Noise words are another meaningless distinction.**
  - **They are redundant.**

Product;
ProductInfo;
ProductData;

Name;
NameString;

Make Meaningful Distinctions

- **Noise words are another meaningless distinction.**
  - **They are redundant.**

Product;
~~***ProductInfo;***~~
~~***ProductData;***~~

Name;
~~***NameString;***~~

Info and Data are indistinct noise like a, an, and the.

Would a name ever be a floating point number?

Use Pronounceable Names

- Software development is a technical and social activity
  - Collaboration among developers
  - Communication with stakeholders

```
class DtaRcrd102
{
        private Date genymdhms;
        private Date modymdhms;
        private final String pszqint = “102”;
        /*…*/
}
```

Use Pronounceable Names

- Humans are good at words. Any words are, by definition, pronounceable.
- If you can't pronounce it, you can't discuss it without sounding like an idiot.

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

Use Pronounceable Names

- Humans are good at words. Any words are, by definition, pronounceable.
- If you can't pronounce it, you can't discuss it without sounding like an idiot.

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

Use Searchable Names

- Single letters are not easy to locate across a body of text.
  - Single-letter names are only preferred to be used as local variables inside short methods.
- The length of a name should correspond to the size of its scope.

i, j, k, n;
MAX_CLASSES_PER_STUDENT;

Naming Conventions

- Classes and objects should have **noun or noun phrase names**:
  - e.g. Customer, WikiPage, Account, and AddressParser
- Methods should have **verb or verb phase names** like:
  - e.g. postPayment, deletePage, or save.
- How about attribute names defined in a class?

Keep Functions Small

Listing 3-1
HtmlUtil.java (FitNesse 20070619)

```
public static String testableHtml(
PageData pageData,
boolean includeSuiteSetup
) throws Exception {
    WikiPage wikiPage = pageData.getWikiPage();
    StringBuffer buffer = new StringBuffer();
    if (pageData.hasAttribute("Test")) {
        if (includeSuiteSetup) {
            WikiPage suiteSetup = PageCrawlerImpl.getInheritedPage(SuiteResponder.SUITE_SETUP_NAME, wikiPage);
            if (suiteSetup != null) {
                WikiPagePath pagePath = suiteSetup.getPageCrawler().getFullPath(suiteSetup);
                String pagePathName = PathParser.render(pagePath);
                buffer.append("!include -setup .")
                    .append(pagePathName)
                    .append("\n");
            }
        }
        WikiPage setup = PageCrawlerImpl.getInheritedPage("SetUp", wikiPage);
        if (setup != null) {
            WikiPagePath setupPath = wikiPage.getPageCrawler().getFullPath(setup);
            String setupPathName = PathParser.render(setupPath);
            buffer.append("!include -setup .")
                .append(setupPathName)
                .append("\n");
        }
    }
    buffer.append(pageData.getContent());
    if (pageData.hasAttribute("Test")) {
        WikiPage teardown = PageCrawlerImpl.getInheritedPage("TearDown", wikiPage);
        if (teardown != null) {
            WikiPagePath tearDownPath = wikiPage.getPageCrawler().getFullPath(teardown);
            String tearDownPathName = PathParser.render(tearDownPath);
            buffer.append("\n")
                .append("!include -teardown .")
                .append(tearDownPathName)
                .append("\n");
        }
        if (includeSuiteSetup) {
            WikiPage suiteTeardown =  PageCrawlerImpl.getInheritedPage(SuiteResponder.SUITE_TEARDOWN_NAME, wikiPage);
            if (suiteTeardown != null) {
                WikiPagePath pagePath = suiteTeardown.getPageCrawler().getFullPath (suiteTeardown);
                String pagePathName = PathParser.render(pagePath);
                buffer.append("!include -teardown .")
                    .append(pagePathName)
                    .append("\n");
            }
        }
    }
    pageData.setContent(buffer.toString());
    return pageData.getHtml();
}
```

Keep Functions Small

Listing 3-2
HtmlUtil.java (refactored)

```
public static String renderPageWithSetupsAndTeardowns(
PageData pageData, boolean isSuite
) throws Exception {
    boolean isTestPage = pageData.hasAttribute("Test");
    if (isTestPage) {
        WikiPage testPage = pageData.getWikiPage();
        StringBuffer newPageContent = new StringBuffer();
        includeSetupPages(testPage, newPageContent, isSuite);
        newPageContent.append(pageData.getContent());
        includeTeardownPages(testPage, newPageContent, isSuite);
        pageData.setContent(newPageContent.toString());
    }
return pageData.getHtml();
}
```

Listing 3-1
HtmlUtil.java (FitNesse 20070619)

```
public static String testableHtml(
PageData pageData,
boolean includeSuiteSetup
) throws Exception {
    WikiPage wikiPage = pageData.getWikiPage();
    StringBuffer buffer = new StringBuffer();
    if (pageData.hasAttribute("Test")) {
        if (includeSuiteSetup) {
            WikiPage suiteSetup = PageCrawlerImpl.getInheritedPage(SuiteResponder.SUITE_SETUP_NAME, wikiPage);
            if (suiteSetup != null) {
                WikiPagePath pagePath = suiteSetup.getPageCrawler().getFullPath(suiteSetup);
                String pagePathName = PathParser.render(pagePath);
                buffer.append("!include -setup .")
                    .append(pagePathName)
                    .append("\n");
            }
        }
        WikiPage setup = PageCrawlerImpl.getInheritedPage("SetUp", wikiPage);
        if (setup != null) {
            WikiPagePath setupPath = wikiPage.getPageCrawler().getFullPath(setup);
            String setupPathName = PathParser.render(setupPath);
            buffer.append("!include -setup .")
                .append(setupPathName)
                .append("\n");
        }
    }
    buffer.append(pageData.getContent());
    if (pageData.hasAttribute("Test")) {
        WikiPage teardown = PageCrawlerImpl.getInheritedPage("TearDown", wikiPage);
        if (teardown != null) {
            WikiPagePath tearDownPath = wikiPage.getPageCrawler().getFullPath(teardown);
            String tearDownPathName = PathParser.render(tearDownPath);
            buffer.append("\n")
                .append("!include -teardown .")
                .append(tearDownPathName)
                .append("\n");
        }
        if (includeSuiteSetup) {
            WikiPage suiteTeardown =  PageCrawlerImpl.getInheritedPage(SuiteResponder.SUITE_TEARDOWN_NAME, wikiPage);
            if (suiteTeardown != null) {
                WikiPagePath pagePath = suiteTeardown.getPageCrawler().getFullPath (suiteTeardown);
                String pagePathName = PathParser.render(pagePath);
                buffer.append("!include -teardown .")
                    .append(pagePathName)
                    .append("\n");
            }
        }
    }
    pageData.setContent(buffer.toString());
    return pageData.getHtml();
}
```

Keep Functions Small: Rules to Remember

- The first rule of functions is that they should be small!
  - **Functions should do one thing. Do only one thing and do it well, i.e. no side effects**
- Use descriptive names. Don’t be afraid to make a name long. A long descriptive name is better than a short enigmatic name.
- Three arguments should be avoided where possible.
  - More than three requires very special justification—and then shouldn’t be used anyway.
- Don’t repeat yourself (i.e. clone your code)!

Use Comments

- Use comments to make good code better
- Don’t command bad code – rewrite it!!!

Use Comments

- Use comments to make good code better
  - Explain yourself
  - Legal comments
  - Informative comments
  - Explanation of intent
  - Clarification
  - Warning of Consequences
  - To-Do comments
  - Amplification
- Don’t command bad code – rewrite it!!!

Explain Yourself

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

Explain Yourself

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

Or this?

```
if (employee.isEligibleForFullBenefits())
```

Explain Yourself

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

The best comment is your code!!!

```
if (employee.isEligibleForFullBenefits())
```

Legal Comments

- Legal Comments: Sometimes our corporate coding standards force us to write certain comments for legal reasons.
  - For example, copyright and authorship statements are necessary and reasonable things to put into a comment at the start of each source file.

```
// Copyright (C) 2003,2004,2005 by Object Mentor, Inc. All rights reserved.
// Released under the terms of the GNU General Public License version 2 or later.
```

Informative Comments

- It is sometimes useful to provide basic information with a comment.

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

Explanation of Intent

- Sometimes a comment goes beyond just useful information about the implementation and provides the intent behind a decision.

```
public int compareTo(Object o)
{
        if(o instanceof WikiPagePath){
                WikiPagePath p = (WikiPagePath) o;
                String compressedName = StringUtil.join(names, "");
                String compressedArgumentName = StringUtil.join(p.names, "");
                return compressedName.compareTo(compressedArgumentName);
        }
        return 1;
        // we are greater because we are the right type.
}
```

Clarification

- Sometimes it is just helpful to translate the meaning of some obscure argument or return value into something that's readable.

```
public void testCompareTo() throws Exception
{
        WikiPagePath a = PathParser.parse("PageA");
        WikiPagePath ab = PathParser.parse("PageA.PageB");
        WikiPagePath b = PathParser.parse("PageB");
        WikiPagePath aa = PathParser.parse("PageA.PageA");
        WikiPagePath bb = PathParser.parse("PageB.PageB");
        WikiPagePath ba = PathParser.parse("PageB.PageA");
        assertTrue(a.compareTo(a) == 0); // a == a
        assertTrue(a.compareTo(b) != 0); // a != b
        assertTrue(ab.compareTo(ab) == 0); // ab == ab
        assertTrue(a.compareTo(b) == -1); // a < b
        assertTrue(aa.compareTo(ab) == -1); // aa < ab
        assertTrue(ba.compareTo(bb) == -1); // ba < bb
        assertTrue(b.compareTo(a) == 1); // b > a
        assertTrue(ab.compareTo(aa) == 1); // ab > aa
        assertTrue(bb.compareTo(ba) == 1); // bb > ba
}
```

Warning of Consequences

- Sometimes it is useful to warn other programmers about certain consequences.

```
// Don't run unless you have some time to kill.
public void _testWithReallyBigFile()
{
        writeLinesToFile(10000000);
        response.setBody(testFile);
        response.readyToSend(this);
        String responseString = output.toString();
        assertSubString("Content-Length:
        1000000000", responseString);
        assertTrue(bytesSent > 1000000000);
}
```

TODO Comments

- It is sometimes reasonable to leave “To do” notes in the form of //TODO comments.

```
//TODO-MdM these are not needed
// We expect this to go away when we do the checkout model
protected VersionInfo makeVersion() throws Exception
{
        return null;
}
```

Amplification

- A comment may be used to amplify the importance of something that may otherwise seem inconsequential.

```
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()));
```

The Ultimate Key to Clean Code

- The key to clean code is ***craftsmanship***
- You must ***gain the knowledge*** of principles, patterns, practices, and heuristics that a craftsman knows.
- And, you must also grind that knowledge into your fingers, eyes, and gut by ***working hard and practicing***.