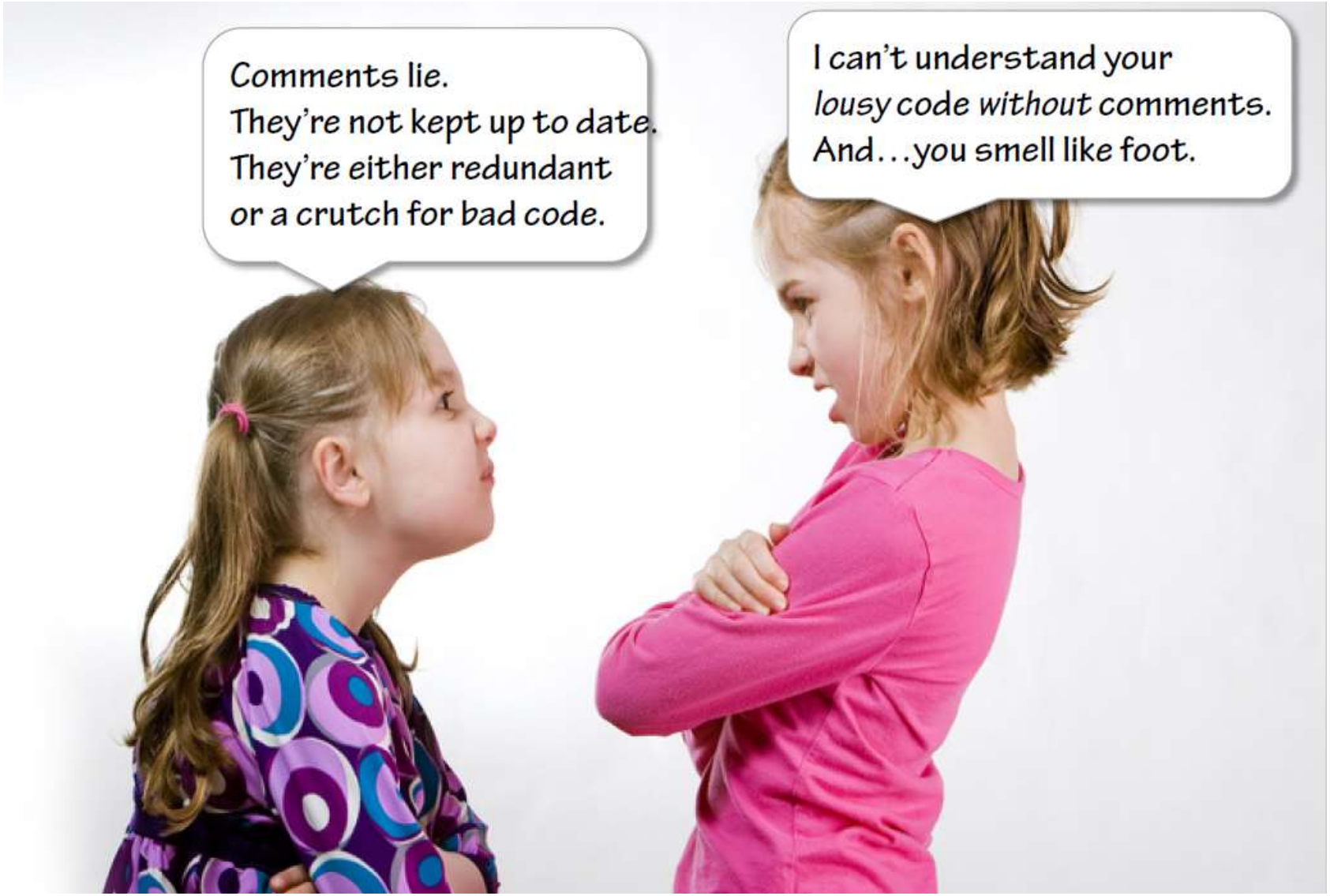
COMMENTS

To comment or not to comment



Comments

- “Purpose of a comment is to explain code that does not explain itself.”
- Comments do not make up for bad code
  - Clear and expressive code with few comments is far superior to cluttered and complex code with lots of comments.
- Don‘t use a comment when you can use a method or a variable
- Comments can contain lies
  - They are not always changed with the code
  - Inaccurate comments are far worse than no comments at all

Comments

- General Rules:
  1. Prefer expressive code over comments.
  2. Use comments when code alone can't be sufficient.
- Comments are useful, but generally a last resort
  - Rather than spend your time writing the comments that explain the mess you've made, spend it cleaning that mess.

Explain yourself in code

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

NOT OK

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

OK

Create a method that says the same thing as the comment you want to write

Comments

- Good comments
  - Legal comments
  - Informative comments
  - Clarification
  - TODO comments
- Bad comments
  - Redundant comments
  - Intent comments
  - Divider comments
  - Mumbling
  - Redundant comments
  - Journal comments
  - Noise comments

Legal comments


```
// 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

OK

```
/// Create an RGB color from three floats in the range 0..1
/// Out of range values will be clamped to this range
RGBColor(float red, float green, float blue);
```

You do not want to add this information to the method name or parameter names

Clarification


```
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
```

Also risky.
Make sure there is no other way
Make sure they are correct

TODO comment


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

Comments

- Good comments
  - Legal comments
  - Informative comments
  - Clarification
  - TODO comments
- Bad comments
  - Redundant comments
  - Intent comments
  - Divider comments
  - Mumbling
  - Redundant comments
  - Journal comments
  - Noise comments

Redundant comments

NOT OK

```
int i = 1; // Set i = 1


var cory = new User(); //Instantiate a new user


/// <summary>
/// Default Constructor
/// </summary>
public User()
{
}


/// <summary>
/// Calcuates Total Charges
/// </summary>
private void CalculateTotalCharges()
{
    //Total charges calculated here
}
```

- Assume your reader can read.
- Don't repeat yourself.

Intent comments

```
// Assure user's account is deactivated.
if (user.Status == 2)
```

NOT OK

```
if (user.Status == Status.Inactive)
{

}
```

OK

Divider comments

NOT OK

```
private void MyLongFunction()
{
    lots
    of
    code

    //Start search for available concert tickets

    lots
    of
    concert
    search
    code

    //End of concert ticket search

    lots
    more
    code
}
```

Mumbling

NOT OK

```
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
}
```

What does this comment mean?

Who loads all the defaults?
When are the defaults loaded?
Why is the catch block empty?

Redundant comments

NOT OK

The comment
- Is not more informative than the code
- Does not provide intent or rationale
- Is not easier to read than the code
- Is less precise than the code

Completely redundant

```
// 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");
  }
}
```

Journal comments

NOT OK

Nowadays we have source control systems
Do not write these 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);
**/
```

Noise comments

NOT OK

```
/**
* 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;
}
```

Closing brace comments

NOT OK

```
public class wc {
  public static void main(String[] args) {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    String line;
    int lineCount = 0;
    int charCount = 0;
    int wordCount = 0;
    try {
      while ((line = in.readLine()) != null) {
        lineCount++;
        charCount += line.length();
        String words[] = line.split("\\W");
        wordCount += words.length;
      } // while
      System.out.println("wordCount = " + wordCount);
      System.out.println("lineCount = " + lineCount);
      System.out.println("charCount = " + charCount);
    } // try
    catch (IOException e) {
      System.err.println("Error:" + e.getMessage());
    } // catch
  } // main
}
```

If you want/need closing brace comments, try to shorten your functions

Commented-out code

NOT OK

```
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()));
```

Why are they commented out?
Are they important?
Can we delete it?

Too much information

NOT OK

```
/*
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.
*/
```

CODE SMELL EXAMPLES

Code smell example

```
double disabilityAmount() {
  if (seniority < 2) {
    return 0;
  }
  if (monthsDisabled > 12) {
    return 0;
  }
  if (isPartTime) {
    return 0;
  }
  // compute the disability amount
  //...
}
```

NOT OK

```
double disabilityAmount() {
  if (isNotEligableForDisability()) {
    return 0;
  }
  // compute the disability amount
  //...
}
```

OK

Code smell example

```
if (date.before(SUMMER_START) || date.after(SUMMER_END)) {
  charge = quantity * winterRate + winterServiceCharge;
}
else {
  charge = quantity * summerRate;
}
```

NOT OK

```
if (isSummer(date)) {
  charge = summerCharge(quantity);
}
else {
  charge = winterCharge(quantity);
}
```

OK

Code smell example

```
void renderBanner() {
  if ((platform.toUpperCase().indexOf("MAC") > -1) &&
       (browser.toUpperCase().indexOf("IE") > -1) &&
        wasInitialized() && resize > 0 )
  {
    // do something
  }
}
```

NOT OK

```
void renderBanner() {
  final boolean isMacOs = platform.toUpperCase().indexOf("MAC") > -1;
  final boolean isIE = browser.toUpperCase().indexOf("IE") > -1;
  final boolean wasResized = resize > 0;

  if (isMacOs && isIE && wasInitialized() && wasResized) {
    // do something
  }
}
```

OK

Code smell example

```
class Order {
  // ...

   public double calculateTotal() {
    double total = 0;
    for (Product product : getProducts()) {
      total += product.quantity * product.price;
    }

    // Apply regional discounts.
    switch (user.getCountry()) {
      case "US": total *= 0.85; break;
      case "RU": total *= 0.75; break;
      case "CN": total *= 0.9; break;
      // ...
    }

    return total;
  }
```

NOT OK

```
class Order {
  // ...

  public double calculateTotal() {
    double total = 0;
    for (Product product : getProducts()) {
      total += product.quantity * product.price;
    }
    total = applyRegionalDiscounts(total);
    return total;
  }

  public double applyRegionalDiscounts(double
                                          total) {
    double result = total;
    switch (user.getCountry()) {
      case "US": result *= 0.85; break;
      case "RU": result *= 0.75; break;
      case "CN": result *= 0.9; break;
      // ...
    }
    return result;
  }
```

OK

Code smell example

```
class Bird {
  //...
  double getSpeed() {
    switch (type) {
      case EUROPEAN:
        return getBaseSpeed();
      case AFRICAN:
        return getBaseSpeed() - getLoadFactor() * numberOfCoconuts;
      case NORWEGIAN_BLUE:
        return (isNailed) ? 0 : getBaseSpeed(voltage);
    }
    throw new RuntimeException("Should be unreachable");
  }
}
```

NOT OK

Code smell example

```
abstract class Bird {
  //...
  abstract double getSpeed();
}
```

```
class European extends Bird {
  double getSpeed() {
    return getBaseSpeed();
  }
}
```

```
class African extends Bird {
  double getSpeed() {
    return getBaseSpeed() - getLoadFactor() * numberOfCoconuts;
  }
}
```

```
class NorwegianBlue extends Bird {
  double getSpeed() {
    return (isNailed) ? 0 : getBaseSpeed(voltage);
  }
}
```

```
// Somewhere in client code
speed = bird.getSpeed();
```

OK