Comments

“Don’t comment bad code-rewrite it.”

Frank Sholey

Comments Do Not Make Up for Bad Code

- A comment will not help out messy, disorganized code.

- Clear and expressive code with few comments

VS

- Cluttered and complex code with lots of comments.

Explainable code

Comment simple 'if' statement

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

Code explains itself

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

Good Comments

- **Legal comments-** copyright and authorship statements
- **Informative comments-** only when necessary
- **Explanation of Intent –** provides the intent behind a decision.
- **Clarification –** translate meaning of some obscure argument or return value into something that is readable.
- **Warning of consequences**
- **TODO comments**- reminders to do something in the future that might not be able to be done right now. Erase after implementing whatever had to be done.
- **Amplification**-amplify importance of something that might not stand out.
- **Javadocs**

Good Comments

Explanation of Intent

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

// You might not agree with this strategy, but at least you know what the author was trying to do.
```

Good Comments

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
```

Good Comments

Warning of Consquences

```
// Don't run this test case 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", resString);
        assertTrue(bytesSent > 1000000000);
}
```

Bad Comments

- **Mumbling** –throwing in a comment just to comment.
- **Redundant Comments** – commenting when the code is already descriptive enough.
- **Misleading Comments** –Inaccurate comments
- **Mandated Comments** – Some companies might have rules that every function or variable must have a comment. Clutters the code.
- **Journal Comments** - Add comment at start of module every time you edit it. Turns into a journal/log of every change.
- **Noise Comments** – restate the obvious and provide no new info.

Bad Comments

- **Attributions and Bylines** /* Added by Rick */
- **Commented-Out Code** – Others who see it won't know if it is important or not, so no one will delete it and it builds up.
- **HTML Comments** –HTML within comments is hard to read
- **Too much information** – too long with unnecessary info
- **Inobvious Connection**- connection between comment and code should be obvious.

Bad Comments

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 = duration;
    cdList.add(cd);
}
```

Bad Comments

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);
* 27-Aug-2002 :
Fixed bug in addMonths() method, thanks to N???levka Petr (DG);
* 03-Oct-2002 : Fixed errors reported by Checkstyle (DG);
* 13-Mar-2003 : Implemented Serializable (DG);
* 29-May-2003 : Fixed bug in addMonths method (DG);
* 04-Sep-2003 : Implemented Comparable. Updated the isInRange javadocs (DG);
* 05-Jan-2005 : Fixed bug in addYears() method (1096282) (DG);
```

Good and Bad

Closing Brace Comments

- Very helpful for long functions with deeply nested structures
- Clutters small functions
- Can be redundant

Function Headers

- Great for long or complex functions
- Short simple functions don't need much description
- Well-chosen function name is usually better than a comment header

Conclusion

- Try to have descriptive names to minimize comments.
- Not all comments are good comments.
- Always update comments as the code updates.
- Keep comments in a high state of repair, relevance, and accuracy.
- But do not focus more on the comments than the code.
- “Don’t comment bad code-rewrite it.”