Mohammadmahdi Farrokhy

Clean Code:

A Handbook of Agile Software Craftsmanship

Chapter 04: Comments

Are comments really that important?

Comments are often unrecognized components of code that can confuse readers and lead to misunderstandings. Bad code should not be commented; it should be refactored. Comments are liars and harm the software. If code requires comments, it may not express the author's intention clearly. Comments are often not updated along with the code. They lead to failure. Codes get refactored, but comments don't. They can become detached from the code they explain.

While I'd like to say *"don't use comments"*, it's not always practical. Because sometimes we are forced to use comments. Instead let's discuss some important points related to commenting.

Points around commenting

- Point 1: Comments Don’t Repay For Bad Code.

If a section of code is unclear, focus on cleaning and improving it rather than just commenting.

Points around commenting

- Point 2: Express Yourself In Code.

Code is the most effective way to communicate the author's message to fellow developers. Embrace this tool rather than resorting to comments.

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

```
if (employees.isEligibleForFullBenefits())
```

Points around commenting

- Point 3: Legal Comments Are Acceptable.

Using copyright comments is permissible.

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

Points around commenting

- Point 4: Informative Comments For Abstractions Are Acceptable.

It's helpful to use a comment for abstracted methods or classes.

```
// Returns an instance of the Responder being tested
protected abstract Responder responderInstance();
```

Points around commenting

- Point 5: Explanation Of An Ambiguous Code By Comment Is Valuable.

Explaining the reasoning behind a particular code decision.

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

Points around commenting

- Point 6: Clarification Of An Ambiguous Code By Comment Is Valuable.

If the method is part of an standard library and can not be cleaned by the programmer, it's a good idea to clarify the code using comments.

```
assertTrue(a.compareTo(b) != 0); // a != b
assertTrue(ab.compareTo(ba) == 0); // ab == ba
```

Points around commenting

- Point 7: Offering Warnings Through Comments Is Useful.

Notify others if certain code sections might be time-consuming to run.

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

Points around commenting

- Point 8: TODO Comments Are Beneficial.

They serve as reminders for future tasks.

```
//TODO: This method should be deleted
protected VersionInfo makeVersion() throws Exception {
    return null;
}
```

Points around commenting

- Point 9: Mumbling Comments Are Bad.

Comments that can not be understood. If you're up to writing comments do your best.

```
public void loadProperties {
    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
    }
}
```

Points around commenting

- Point 10: Redundant Comments Are Bad.

A comment for a simple method is redundant and needs to be deleted.

Points around commenting

- Point 11: Misleading Comments Are Bad.

Sometimes we use words and phrases in our comments that do not send the correct message from us to the reader. It may lead to misunderstanding of the code's intention.

Points around commenting

- Point 12: Forced Comments Are Bad.

For example commenting every argument in a method.

Points around commenting

- Point 13: Journal Comments Are Outdated.

Recording every code change with dates in comments is outdated, as source control software now handles this.

Points around commenting

- Point 14: Marker Comments Are Problematic.

While they might seem helpful initially, they can become misplaced during code changes and cause confusion.

```
private void doSomething(){
    method1();
    method2();

    // Getting The Result
    method3();
    method4();

    // Sorting The Result
    method5();
    method6();
}
```

Points around commenting

- Point 15: Author’s Name

IDEs and version control systems provide information on code changes. Commenting the author's name is redundant.

Points around commenting

- Point 16: Commented Codes

It's one of the dirtiest codes to write. A commented code confuses the reader.

- *If this code is redundant why is it commented and not deleted?*
- *Is this code going to change? If so, why is there no TODO comments above it?*
- *What happens if I delete this code?*