Clean Code:

A Handbook of Agile Software Craftsmanship

Chapter 04: Comments

Are comments really that important?

Comments are one of most unrecognized elements of code that could mislead the reader and cause a lot of confusion. Bad code should not be commented; it should be refactored. Comments are liars and damage the software. If a code needs comments, it does not reveal the author's intention. Comments are often not updated along with the code. They lead to failure. Codes get refactored, but comments don't. Comments are often isolated from the code they describe.

I would rather say *"don't use comments"*, but that would be unrealistic. Because sometimes we are forced to use comments. Instead let's talk about the points around commenting.

Points around commenting

- Point 1: Comments don’t repay for bad code.

If you have an unrecognized module clean it, don’t comment it.

Points around commenting

- Point 2: Describe yourself in code.

Code is the best tool to send the author's message to other developers. Don't ignore it and use comment instead of it.

```
// 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 good.

Using copyright comments is not forbidden.

```
// 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 good.

It's a good idea to use a comment for an abstracted method or class.

```
// 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 good.

Talking about the intention behind a decision in the code.

```
// 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 good.

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: Giving warnings by comment is good.

Some parts of code may be time consuming to run. We can alert the others using comments.

```
// 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 good.

It is a reminder for a later task to do.

```
//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: Extra comments are bad.

A comment for a simple method is extra 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 bad.

Commenting every change in the code with the date makes the code so unrecognized. Now the source control software take care of it.

Points around commenting

- Point 14: Marker comments are bad.

They may be helpful at the first look. But when the code changes they may be in a wrong location and create confusion.

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

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

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

Points around commenting

- Point 14: Name of the author

IDEs and version controllers can tell us who changed the line of code. So commenting the name of the author or editor is a really bad idea.

Points around commenting

- Point 15: 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?*