Mohammadmahdi Farrokhy

Clean Code:

A Handbook of Agile Software Craftsmanship

Chapter 04: Comments

Are comments really that bad?

- Comments are often unrecognized components of code that can confuse readers and lead to misunderstanding.
- Comments are used to cover for bad code, instead of fixing it.
- If code requires comments, it means, it can not express the author's intention clearly.
- Comments are often not updated along with the code.
- Codes get refactored, but comments don't.
- Comments might become detached from the code they explain.

Although I'd like to say ***"don't use comments"***, it's not always practical. Because sometimes they can lead to value. So we should consider some points about them.

Points around commenting

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

If a section of code is unclear, don't comment it, Refactor it.

Points around commenting

- Point 2: Express Yourself In Code.

Code is the most effective way to transfer the author's message to fellow developers.

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

✗

```
// GOOD CODE
if(employees.isEligibleForFullBenefits())
```

✓

Points around commenting

- Point 3: Legal Comments Are Acceptable.

Copyright comments could be used.

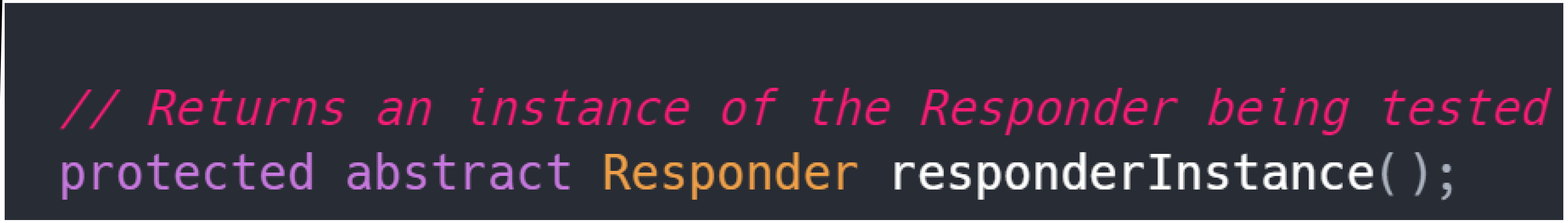
```
// Copyright (C) 2003, 2004, 2005 by Object Mentor, Inc. All Rights reserved.
// Release under the terms of the GNU General Public License version 2 or later
```

✓

Points around commenting

- Point 4: Informative Comments For Abstractions Are Acceptable.

It's helpful to use a comment for abstract 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 reason behind a particular code decision.

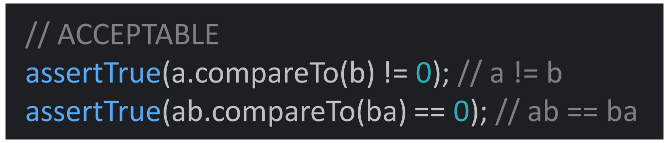
```
// ACCEPTABLE
// This is our best attempt to get a race condition
// by creating large number of threads.
for (int i = 0; i< 25000; i++)
{
   WidgetBuilderThread widgetBuilderThread = new WidgetBuilderThread(widgetBuilder, text, parent, failFlag);
   Thread thread = new Thread(widgetBuilderThread);
   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 refactored by the programmer, it's a good idea to make it clear.



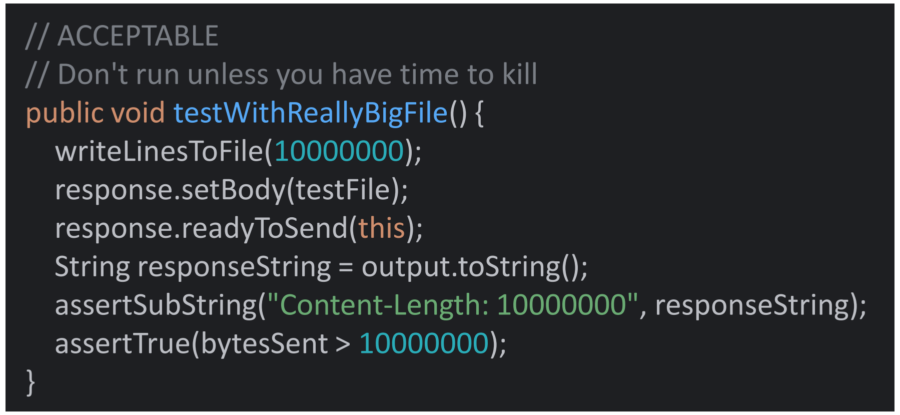
```
// ACCEPTABLE
assertTrue(a.compareTo(b) != 0); // a != b
assertTrue(ab.compareTo(ba) == 0); // ab == ba
```

✓

Points around commenting

- Point 7: Offering Warnings Through Comments Is Useful.

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



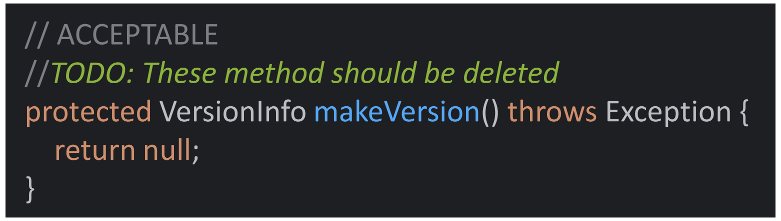
```
// ACCEPTABLE
// Don't run unless you have time to kill
public void testWithReallyBigFile() {
    writeLinesToFile(10000000);
    response.setBody(testFile);
    response.readyToSend(this);
    String responseString = output.toString();
    assertSubString("Content-Length: 10000000", responseString);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

✓

Points around commenting

- Point 8: TODO Comments Are Beneficial.

They serve as reminders for future tasks.



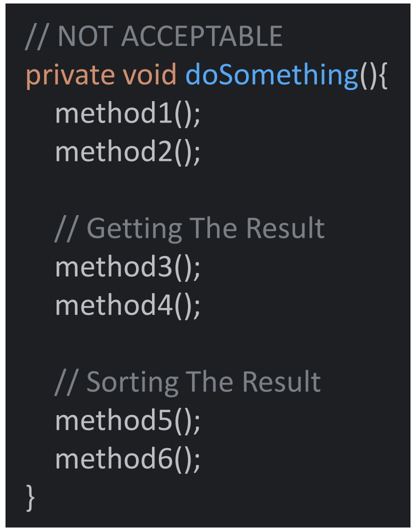
```
// ACCEPTABLE
//TODO: These method should be deleted
protected VersionInfo makeVersion() throws Exception {
   return null;
}
```

✓

Points around commenting

- Point 9: Marker Comments Are Problematic.

While they might seem helpful in the first look, they can become misplaced during code changes and cause confusion.



```
// NOT ACCEPTABLE
private void doSomething(){
    method1();
    method2();

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

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

❌

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 to the reader. It may lead to misunderstanding of the code's intention.

Points around commenting

- Point 12: Obsessive Comments Are Bad.

Like 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: Author's Name Is Not Acceptable

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

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 be used later? If so, why is there no TODO comments around it?*
- *What happens if I delete this code?*