Chapter 4 - Comments

Nothing can be quite so helpful as a well-placed comment. Nothing can clutter up a module more than frivolous dogmatic comments. Nothing can be quite so damaging as an old comment that propagates lies and misinformation.

If our programming languages were expressive enough, or if we had the talent to subtly wield those languages to express our intent, we would not need comments very much—perhaps not at all.

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. 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))
```

vs

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

Good Comments

Some comments are necessary or beneficial. However the only truly good comment is the comment you found a way not to write.

Legal Comments

Sometimes our corporate coding standards force us to write certain comments for legal reasons. For example, copyright and authorship statements are necessary and reasonable things to put into a comment at the start of each source file.

Informative Comments

It is sometimes useful to provide basic information with a comment. For example, consider this comment that explains the return value of an abstract method:

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

A comment like this can sometimes be useful, but it is better to use the name of the function to convey the information where possible. For example, in this case the comment could be made redundant by renaming the function: `responderBeingTested`.

Explanation of Intent

Sometimes a comment goes beyond just useful information about the implementation and provides the intent behind a decision. Example:

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

Clarification

Sometimes it is just helpful to translate the meaning of some obscure argument or return value into something that's readable. In general it is better to find a way to make that argument or return value clear in its own right; but when its part of the standard library, or in code that you cannot alter, then a helpful clarifying comment can be useful.

Warning of concequences

Sometimes it is useful to warn other programmers about certain consequences.

```
// Don't run 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", responseString);
  assertTrue(bytesSent > 1000000000);
}
```

TODO Comments

It is sometimes reasonable to leave “To do” notes in the form of //TODO comments. In the following case, the TODO comment explains why the function has a degenerate implementation and what that function's future should be.

```
//TODO-MdM these are not needed
// We expect this to go away when we do the checkout model
```

```
protected VersionInfo makeVersion() throws Exception {
  return null;
}
```
TODOs are jobs that the programmer thinks should be done, but for some reason can't do at the moment. It might be a reminder to delete a deprecated feature or a plea for someone else to look at a problem. It might be a request for someone else to think of a better name or a reminder to make a change that is dependent on a planned event. Whatever else a TODO might be, it is not an excuse to leave bad code in the system.

Amplification

A comment may be used to amplify the importance of something that may otherwise seem inconsequential.

```
String listItemContent = match.group(3).trim();
// the trim is real important. It removes the starting
// spaces that could cause the item to be recognized
// as another list.
new ListItemWidget(this, listItemContent, this.level + 1);
return buildList(text.substring(match.end()));
```

Javadocs in Public APIs

There is nothing quite so helpful and satisfying as a well-described public API. The javadocs for the standard Java library are a case in point. It would be difficult, at best, to write Java programs without them.

Bad Comments

Most comments fall into this category. Usually they are crutches or excuses for poor code or justifications for insufficient decisions, amounting to little more than the programmer talking to himself.

Mumbling

Plopping in a comment just because you feel you should or because the process requires it, is a hack. If you decide to write a comment, then spend the time necessary to make sure it is the best comment you can write. Example:

```
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
  }
```

```
}
```

What does that comment in the catch block mean? Clearly meant something to the author, but the meaning not come though all that well. Apparently, if we get an `IOException`, it means that there was no properties file; and in that case all the defaults are loaded. But who loads all the defaults?

Redundant Comments

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

What purpose does this comment serve? It's certainly not more informative than the code. It does not justify the code, or provide intent or rationale. It is not easier to read than the code. Indeed, it is less precise than the code and entices the reader to accept that lack of precision in lieu of true understanding.

Misleading comments

Sometimes, with all the best intentions, a programmer makes a statement in his comments that isn't precise enough to be accurate. Consider for another moment the example of the previous section. The method does not return when `this.closed` becomes `true`. It returns if `this.closed` is `true`; otherwise, it waits for a blind time-out and then throws an exception if `this.closed` is still not true.

Mandated Comments

It is just plain silly to have a rule that says that every function must have a javadoc, or every variable must have a comment. Comments like this just clutter up the code, propagate lies, and lend to general confusion and disorganization.

```
/**
*
* @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);
}
```

Journal Comments

Sometimes people add a comment to the start of a module every time they edit it. Example:

```
* 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);
```

Today we have source code control systems, we don't need this type of logs.

Noise Comments

The comments in the follow examples doesn't provides new information.

```
/**
* Default constructor.
*/
protected AnnualDateRule() {
}
/** The day of the month. */
private int dayOfMonth;
```

Javadocs comments could enter in this category. Many times they are just redundant noisy comments written out of some misplaced desire to provide documentation.

Don't Use a Comment When You Can Use a Function or a Variable

Example:

```
// does the module from the global list <mod> depend on the
// subsystem we are part of?
if (smodule.getDependSubsystems().contains(subSysMod.getSubSystem()))
```

vs

```
ArrayList moduleDependees = smodule.getDependSubsystems();
String ourSubSystem = subSysMod.getSubSystem();
if (moduleDependees.contains(ourSubSystem))
```

Position Markers

This type of comments are noising

```
// Actions ////////////////////////////////
```

Closing Brace Comments

Example:

```
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
```

You could break the code in small functions instead to use this type of comments.

Attributions and Bylines

Example:

```
/* Added by Rick */
```

The VCS can manage this information instead.

Commented-Out Code

```
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()));
```

If you don't need anymore, please delete it, you can back later with your VCS if you need it again.

HTML Comments

HTML in source code comments is an abomination, as you can tell by reading the code below.

```
/**
* Task to run fit tests.
* This task runs fitnesse tests and publishes the results.
* <p/>
* <pre>
* Usage:
* &lt;taskdef name=&quot;execute-fitnesse-tests&quot;
* classname=&quot;fitnesse.ant.ExecuteFitnesseTestsTask&quot;
```

```
* classpathref=&quot;classpath&quot; /&gt;
* OR
* &lt;taskdef classpathref=&quot;classpath&quot;
* resource=&quot;tasks.properties&quot; /&gt;
* <p/>
* &lt;execute-fitnesse-tests
* suitepage=&quot;FitNesse.SuiteAcceptanceTests&quot;
* fitnesseport=&quot;8082&quot;
* resultsdir=&quot;${results.dir}&quot;
* resultshtmlpage=&quot;fit-results.html&quot;
* classpathref=&quot;classpath&quot; /&gt;
* </pre>
*/
```

Nonlocal Information

If you must write a comment, then make sure it describes the code it appears near. Don't offer systemwide information in the context of a local comment.

Too Much Information

Don't put interesting historical discussions or irrelevant descriptions of details into your comments.

Inobvious Connection

The connection between a comment and the code it describes should be obvious. If you are going to the trouble to write a comment, then at least you'd like the reader to be able to look at the comment and the code and understand what the comment is talking about

Function Headers

Short functions don’t need much description. A well-chosen name for a small function that does one thing is usually better than a comment header.

Javadocs in Nonpublic Code

Javadocs are for public APIs, in nonpublic code could be a distraction more than a help.