CODING HORROR

24 Jul 2008

Coding Without Comments

https://blog.codinghorror.com/coding-without-comments/

If peppering your code with lots of comments is good, then having **zillions of comments** in your code must be *great*, right? Not quite. Excess is one way good comments go bad:

```
'************************************************
' Name: CopyString
'
' Purpose: This routine copies a string from the source
' string (source) to the target string (target).
'
' Algorithm: It gets the length of "source" and then copies each
' character, one at a time, into "target". It uses
' the loop index as an array index into both "source"
' and "target" and increments the loop/array index
' after each character is copied.
'
' Inputs: input The string to be copied
'
' Outputs: output The string to receive the copy of "input"
'
' Interface Assumptions: None
'
```

```
' Modification History: None
'
' Author: Dwight K. Coder
' Date Created: 10/1/04
' Phone: (555) 222-2255
' SSN: 111-22-3333
' Eye Color: Green
' Maiden Name: None
' Blood Type: AB-
' Mother's Maiden Name: None
' Favorite Car: Pontiac Aztek
' Personalized License Plate: "Tek-ie"
'**********************************************
```

I'm constantly running across comments from developers who don't seem to understand that the code already tells us *how* it works; we need the comments to tell us *why* it works. Code comments are so widely misunderstood and abused that you might find yourself wondering if they're worth using at all. Be careful what you wish for. Here's some code with **no comments whatsoever**:

```
r = n / 2;
while ( abs( r - (n/r) ) > t ) {
r = 0.5 * ( r + (n/r) );
}
System.out.println( "r = " + r );
```

Any idea what that bit of code does? It's perfectly readable, but *what the heck does it do?*

Let's add a comment.

```
// square root of n with Newton-Raphson approximation
r = n / 2;
while ( abs( r - (n/r) ) > t ) {
r = 0.5 * ( r + (n/r) );
}
```

```
System.out.println( "r = " + r );
```

That must be what I was getting at, right? Some sort of pleasant, middle-of-the-road compromise between the two polar extremes of no comments whatsoever and carefully formatted epic poems every second line of code?

Not exactly. Rather than add a comment, I'd refactor to this:

```
private double SquareRootApproximation(n) {
r = n / 2;
while ( abs( r - (n/r) ) > t ) {
r = 0.5 * ( r + (n/r) );
}
return r;
}
System.out.println( "r = " + SquareRootApproximation(r) );
```

I haven't added a single comment, and yet this mysterious bit of code is now perfectly understandable.

While comments are neither inherently good or bad, they are frequently used as a crutch. **You should always write your code as if comments didn't exist.** This *forces* you to write your code in the simplest, plainest, most self-documenting way you can humanly come up with.

When you've rewritten, refactored, and rearchitected your code a dozen times to make it easy for your fellow developers to read and understand -- when you can't possibly imagine any conceivable way your code could be changed to become more straightforward and obvious -- then, and *only then*, should you feel compelled to add a comment explaining what your code does.

As Steve points out, this is one key difference between junior and senior developers:

In the old days, seeing too much code at once quite frankly exceeded my complexity threshold, and when I had to work with it I'd typically try to rewrite it or at least comment it heavily. Today, however, I just slog through it without complaining (much). When I have a specific goal in mind and a complicated piece of code to write, I spend my time making it happen rather than telling myself stories about it [in comments].

Junior developers rely on comments to tell the story when they should be relying on the *code* to tell the story. Comments are narrative asides; important in their own way, but in no way meant to replace plot, characterization, and setting.

Perhaps that's the dirty little secret of code comments: **to write good comments you have to be a good writer**. Comments aren't code meant for the compiler, they're words meant to communicate ideas to other human beings. While I do (mostly) love my fellow programmers, I can't say that effective communication with other human beings is exactly our strong suit. I've seen three-paragraph emails from developers on my teams that practically melted my brain. *These* are the people we're trusting to write clear, understandable comments in our code? I think maybe some of us might be better off sticking to our strengths -- that is, writing for the compiler, in as clear a way as we possibly can, and reaching for the comments only as a method of last resort.

Writing good, meaningful comments is hard. It's as much an art as writing the code itself; maybe even more so. As Sammy Larbi said in Common Excuses Used To Comment Code, if your feel your code is too complex to understand *without* comments, **your code is probably just bad.** Rewrite it until it doesn't need comments any more. If, at the end of that effort, you still feel comments are necessary, then by all means, add comments. Carefully.

NEXT

Understanding The Hardware

PREVIOUS

Building Tiny, Ultra Low Power PCs

Written by Jeff Atwood

Indoor enthusiast. Co-founder of Stack Overflow and Discourse. Disclaimer: I have no idea what I'm talking about. Find me here: http://twitter.com/codinghorror

I agree! Comments in code are a double edged sword; they're become like more code to maintain. And if no one is maintaining them, they become downright dangerous.

I use comments sparingly, and only to tell what's being done and how. Usually in the form of one-liners.

Jul '08

Calvin

what the hell is a comment?

Jul '08

Mike

And if no one is maintaining them, they become downright dangerous.

That's just job security. I'll do one better.. insert lies and comments that make no absolutely no sense.

Relax, I kid.. I kid..

Jul '08

LucasG

I've found, when looking back at really old work, that the comments were nothing more then psuedo-code typed before actually making things work.

When I look at my systems today, about the only comments I find (other then TODO stuff) are notes about the business rules being implemented by the code and even then only when the business rules are intuitively obvious.

For instance, in my current project I need week numbers. However, we base week numbers by starting from the last monday of the year before. That's weird enough to actually document as a method named GetWeekNumberStartingFromLastMondayOfPreviousYear is just silly.

I've heard arguments for using comments during maintinance to show what was tried, but frankly, if you're checking in your work on a regular basis, you already have that history.

Jul '08

f0dder

Hm, you should still have a comment in that code snippet stating it uses Newton-Raphson approximation, IMHO. And you definitely don't want that as part of the function name. Heck, even having Approximation in the function name might be putting too much implementation detail in the function name, unless you're dealing with a common code library that just *might* be picked up by engineering.

Oh, and you'd definitely also want some error margin information as part of the function comment...

in case somebody from engineering even *considers* using the routine 

Jul '08

alex13

```
r = n / 2;

while ( abs( r - (n/r) )  t ) {
```

```
  r = 0.5 * ( r + (n/r) );

}

System.out.println( r =  + r );
```

Isn't (n/r) always 2?

1 reply

Jul '08

John_Levon

never, ever start with comments.

Unless it's inline extractable API docs like javadoc of pydoc, of course.

Jul '08

Michael

One problem with your example is that it's taken from numerical analysis, where knowing HOW the computations are made is essential to judging their accuracy for various inputs. Saying This subroutine computes X doesn't convince anybody that it's accurate.

Which brings me to your square-root example. Your final version tells me that the code approximates the square root. However, without a comment to tell me that Newton's method is being used, I cannot know this without spending time examining the code closely and having seen Newton's method in an algorithm before. Most developers would give up well before they make that realization --- forcing them to ask you to explain the algorithm and convince them of its correctness, which wastes more of your time than it would take to write the comment.

1 reply

Jul '08

Flanagan

The only problem with writing understandable code is that many programmers have no idea how confusing their stuff is. Sure, it's 20 pages of unorganized cryptic statements, but THEY understood it. Surely if someone else doesn't understand it, they must be lousy programmmers.

Jul '08

BOB7

I find that when I stub out a method, I pseudocode it first with comments. This helps me think through the logic and scope of the task and generates a roadmap that I can return to if I'm interrupted.

It seems to generate very useful comments for others to follow later.

Jul '08

Nick

never, ever start with comments

I find I write my code cleanest when I start by filling in my methods with what they ought to do in comments, then filling in what they actually do after I've got my architecture down.

And, if I comment in the API doc style of the language, I get nice documentation to boot, as mentioned before.

Jul '08

Gwyn

I couldn't agree more. When I was a junior developer I used to bemoan the lack of comments in code created by my more senior team members. Now that I am a lot more experienced, a lot of comments tend to clutter up the code and reduce the comprehensibility.

Generally, comments tend to be wrong, or confusing. In addition, there is a tendency to comment code that is changing, or complex. When that code is modified further, the comment becomes out of date, but the instinct is to leave the comment in just in case. Before long, nobody knows what a comment means, or why it could possibly be there.

The best comments are like the ones shown in the article. Short and to the point. If you are implementing a particular algorithm, then say so (perhaps with a link to an article explaining the method), but don't overload the reader with useless detail that is covered in the code.

Jul '08

Dan

I think comments are important because most of the programmers actually out there working are pretty hopeless. I may be able to write well-structured, clean code but the rest of my team won't understand it because they're very inexperienced. However, they're all we have.

The same happened to me when I tried to open up DNN when I was new to ASP.net. Beautifully written code, not a single comment, full of bugs. I couldn't make head or tail of it.

Leave the comments on for all the people who aren't brilliant, tireless programers.

Jul '08

Wilson

I will side with the people who say that badly maintained comments are worse than no comments. Comments in the code *are* code and need to be maintained as such; but there's no compiler or interpreter to tell you about the dependencies.

My own code tends to be tersely commented at best; the one time I remember writing about 25 lines of comment for three lines of code was because those three lines were getting around a bug in Windows 3.1 and there was no URL I could point people at to read about it...

Jul '08

Brennan

Most of my comments contain the url from where I stole the particular bit of code from. Tests explain things much better than comments can. Talk is cheap. Show me the code.

Jul '08

codinghorror

Just as others, who have already mentioned it, I too write comments before writing a line of code, as a way of helping me think things through; and I see nothing wrong with that.

Not quite the same thing, unless those comments stay in the code after you've written it!

Jul '08

codinghorror

M Michael:

Your final version tells me that the code approximates the square root. However, without a comment to tell me that Newton's method is being used, I cannot know this without spending time examining the code closely and having seen Newton's method in an algorithm before.

SquareRootApproximation(n, Type.NewtonRaphson)

1 reply

Jul '08

BugRree

You can save on writing comments by designing every piece of software around a concept of software factories vomiting objects but looking really understandable, by-the-book academically cool.

Or you can write compact code, based on well thought-out usage patterns, not some over-engineered ego-busting monster that compiles, runs but cannot be reused.

Jul '08

Barry_Kelly

Key problem: some comments can't be expressed through functional decomposition.

I agree that algorithm and data structure explanation should be done largely through functional and OO decomposition and member and type naming, and that this is one of the big reasons why naming is so important, but there are other aspects to programming than call-tree decomposable algorithms and trivially modular data structures. Not all problem domains and corresponding solution abstractions are representable in a first-class way using the tools of structured programming and object-oriented design.

What about global* state changes that affect horizontal sections of code, rather than the vertically-oriented call tree of a functional decomposition?

(* or local to a large object graph - functionally not all that different)

What about concurrent programming, where every cross-thread-visible operation is only correct when every possible relevant state of every other thread has been considered?

What about older programs written in older languages using older techniques - I'm thinking of IDE compilers written in C with extensive global state, discriminated unions and semantic field reuse across different stages, in particular (since it's my day job). A few comments go a long way in these kinds of situations - particularly since one doesn't have the luxury of writing it correctly the first time, even presuming that the right abstractions are available in any single programming language.

Code that has been maintained for decades generally doesn't have an immediately understandable gestalt such that it can be immediately refactored and rewritten more clearly, and in these situations comments are vital to help build up that gestalt, so that future refactorings can occur.

Motivating example: you've got a data structure graph that represents a parsed piece of code, and you've got to do manipulations, optimizations, debug info extraction, and code generation - and do it all as fast as possible with minimal copying and reallocation etc. Classes don't sit well with the requirements - you want to separate out your parsing logic, debug info, optimization, code generation, etc., rather than lumping them all into a single class. Each of those phases will want to annotate the graph for its own purposes, but those annotations can go away when the phase is done. You end up using discriminated unions or something similar, along with functional pattern matching or a visitor pattern, etc.

I can assure you that that code (and data structure types) will improve with the addition of a few comments 

Jul '08

j1120

I'll remember to do this but perl's plain old documentation is so cool.

Jul '08

Brian

I too like to start with comments if I'm writing a new function that I'm not quite sure about. Otherwise I agree.

The other real problem I see is when people don't keep the comments up-to-date with the rest of the code. As the 'what' changes, the 'why' can change as well. It's frustrating when the code says one thing and the comments say another. Which one is right?

And if the code evolves in a way in which it becomes self-explanatory, then by all means, take the comments out.

Jul '08

Sean

I wish someone would tell this to my High School and University programming teachers...

Jul '08

RobertR

Then the .NET Framework code must be bad because I have to comment my code just to document how the framework works. Why can't we just go back to COBOL? That code was perfectly readable.

Jul '08

keitht1

I think there's something to be said for starting with comments. McConnell gives a thorough explanation of why you should, in fact, start with comments before even writing code in chapter 9 of Code Complete 2.

In short, you start by roughing out the design of the function/method/whatever in pseudocode. Then you rework and refactor the pseudocode until it's as plain as day what you're doing. Only then do you

write the code needed to implement the pseudocode. At the end of the process you've still got the pseudocode there to act as comments.

Why delete it? It certainly helps maintenance developers and is a great check to ensure you actually coded the routine you intended to code.

Anyhow, this is just a thumbnail sketch of why you should use comments. For a much better (and more persuasive) argument, check out McConnell's book. Every professional developer should probably already have a copy anyhow.

Jul '08

Paul_Ritter

I always try to incorporate both ideas in my code. For example, I would write the square root approximation function as

```
SquareRootApproximation(n)

//Newton-Raphson approximation

{

...

}
```

1 reply

Jul '08

DavidA

 codinghorror:

SquareRootApproximation(n, Type.NewtonRaphson)

That's just silly, Jeff. You are going to parameterize the algorithm in order to avoid adding one simple comment? You suggest making the function's interface - and possibly it's implementation - more complex to avoid a comment? That's what I'd call a coding horror.

I have to agree with Jason Bunting. Either that, or you are trying too hard to be controversial and drive traffic to your site.

Jul '08

astigmatik

I think to be more precise, your statement should be: comments must reflect and explain the code exactly, all the time.

I don't care going through a very long comment IF IT HELPS me in the end.

It's not wrong to start with comments. If you can't put to paper what you intend to do, how are you going to start coding?

Jul '08

Nick

A time when comments are a great and wonderful thing:

Given

1. Code you didn't write, but need to visit briefly to fix a problem
2. You spend more time than seemed reasonable to figure out what the hell the code was doing, and it wasn't obvious

Drop a line of comment in with your discoveries so that the next poor sod who has to visit that code won't waste an hour of THEIR life figuring out what the heck is going on.

{ Before anyone mentions that it would be better to fix the crap: 1) in real life, you don't always have time to change code to be perfect, and 2) if you do have the time, rewriting code to satisfy an ideal isn't the best use of the limited number of hours in one's life, unless it will somehow recover some of those hours at a future date }

Jul '08

JeremyP

I'm going to have to disagree with you too, Jeff. Commenting before you code a method body serves to let you flesh out the method before you spend the time to write the code for it. This ensures that you write what you intended to write, as you have the comment to look back to if you get stuck. Extracting API documentation from your code comments is a beautiful, beautiful thing. That means actually putting API documentation comments in your code.

In a previous entry you complained about the verbosity of code, but today you're advocating it in order to reduce unnecessary comments? Something doesn't quite add up.

Jul '08

SamratP

@fodder
i agree with you...

but may be because im just a junior developer...and im not being sarcastic

may be experience will make me realize what Jeff and Steve are trying to say out here...mind you Jeff's SquareRootApproximation(n, Type.NewtonRaphson) is very legible, but there are times when self explanatory function names are not possible. It just saves me a lot of *slog* with some intelligent comments(or for that matter, even some unintelligent ones)peppered here and there.

Selecting a lighter shade for comments in the IDE makes life a lot easier for me(replacing default dark green in VS2005 to a light blue) to avoid those huge huge blocks of comments from bugging you.

Jeff may be you post one on IDE color scheme. Since your post on consolas font has helped me a lot.

Jul '08

Will

I have to agree with David Avraamides up there about your approximation code.

I code primarily in PHP, and my site uses quite a few includes, so I have to comment to remind myself where certain functions/variables which are being called are located.

Jul '08

Steve

It's rather silly to see so many comments against comments...

Well written comments, or even well placed comments, can save lots of work.

Relying on procedure names to comment?

Jul '08

Tim_Yen

I usually use the remarks section in c# to explain the why of a method if required.

Jul '08

Adam

I do not write many comments except for particularly arcane functions and classes. Sometimes, and Im sure Im not alone, I dont know exactly the best way about doing something but I figure a logical way of doing what I need done. It may not be the best (because I Was not aware of a built in function or a method used to arrive at said conclusion), but it works. However, it works in a very difficult to understand manner (well maybe its just me). This is where I throw comments in.

But quite frankly, Ive maintained code that had comments plastered all over the login scripts. Are you kidding me? First year college students understand login scripts, why the hell are people commenting all over the place?

Mostly I avoid comments, but then I dont use stupid variable names like $b (or var b or Dim b or whatever). Of course, I wouldnt use SquareRootApproximation either (probably sqRtAppx which is perfectly understandable to me).

Jul '08

arle_nadja

Too many or wrong comments = bad
so no comments at all = good? What? How about well written and maintained comments that explain what you are doing? (not how you are doing it)

Jul '08

Assaf_Lavie

Comments are important, even though in many cases clearer coding does the trick instead. The idea of comments explaining the why instead of the what is nice, but I think I have an even more convincing argument in favor of comments, which runs as follows.

Code, clear as it may be and with the most elegant of naming conventions applied to it, can only ever document itself. That is, such rare self-explanatory code can only document what's there, never what's not there. What's not there is all the different (usually naive) ways the original coder tried to write the code your reading, that were wrong for some reason. The role of comments is thus to explain those inherently unintuitive cases were code isn't as simple as it might have been in a perfect world. All those hidden pitfalls owing to bugs, support for some legacy design or a 3rd party workaround; all those algorithms that were tried and were found imperfect for the job (that some smarty-pants newcomer may wish to retry on his first day unless warned beforehand); etc. In all these cases, information critical to understanding the what hides in the what not, and can only be exposed by explicit documentation (e.g. comments).

Jul '08

James

I agree with you Jeff. At least that his example:

SquareRootApproximation(n)

is more than adequate. You may argue that if by looking at the code and reading the procedure name you should almost instanly find that you're getting squareroots and using an approximation method. The comment will NOT explain the why unless you write a huge comments that honestly defeats the purpose. If the person reading the code is not current with his/her maths... They'll end up looking it in wikipedia comment or not. The difference is that the people that will see the code at first glance wont lose their time. And the people that don't get it will end up in the same wikipedia page with the comment or with well named objects anyways.

Jul '08

JaredP

I have yet to run into to code which was too commented.

Jul '08

chico

'never, ever start with comments' .. what a load of bollocks. if more people started with comments that outlined what they were attempting to achieve maintenance would be a heck of a lot easier, and so would spotting their bugs.

While I agree that ideally you want to end up with self-documenting code, more often than not you're writing rubbish on the first pass. And, pragmatically speaking, sometimes, a first pass is all that it gets until you have time to go back over it. When you do have time to go over it, those comments might help you or someone else refactor more efficiently.

It seems more often than not that people don't write what they mean to write, so comments should help spot bugs and they act like a checksum. In a code review if someone spots the comments differ from the code, then it can be followed up on.

Or just be like everyone else and write code for yourself with little regard to others. Don't unit test. If it compiles it must works. Phooey.

Just because some people drive like idiots, do you ban cars, or do you punish the idiot?

1 reply

Jul '08

codinghorror

P Paul_Ritter:

SquareRootApproximation(n)
// Newton-Raphson approximation

Sure, this is fine too; I only listed my version as a counterpoint to you must add a comment otherwise nobody will know what approximation you're using.

As I said in the post: If, at the end of that effort, you still feel comments are necessary, then by all means, add comments. Carefully.

C chico:

if more people started with comments that outlined what they were attempting to achieve

Again, this is not at all what the post is about. Unless those comments remain in the code after you've written the function. I'm tempted to delete that sentence because so many people are misunderstanding it.

Jul '08

codinghorror

we need to end this myth that code is EVER as clear as well-written English.

Really? Most of the english I read online is anything but well-written. At least code *compiles* (or interprets, if that's your cup of tea).

Jul '08

T

Tom_Dibble

James -

I disagree. If you are relying on a well-known algorithm, you should identify the specific algorithm. Look up Square Root Approximation in Wikipedia, and I see nothing about the Newton-Raphson method or anything that looks like it will match the code. If I read really carefully, I see a link to Newton's Method, which in fact does match the algorithm here. That's only after wasting a huge amount of time reading other approaches which don't match the given approach.

IMHO, always default to descriptive method names for algorithms, and always include wiki-friendly (or dictionary-friendly, or old-school-encyclopedia-friendly) terms. Therefore:

```
NewtonRaphsonSquareRootApprox(n)
```

Or:

```
/**

 * Newton-Raphson Square Root Approximation

 */

NRSquareRootApprox()
```

Some purists will say too much implementation detail, but IMHO you want that implementation detail in there. If you change the implementation to a different algorithm, it is right and proper that you should have to visit every single use of the method in your codebase to make sure the new algorithm will work as well there as the one you are targetting. And, if it doesn't, you then end up with NRSquareRootApprox() and ISRSquareRootApprox when you add in an Integer Square Root method for greater speed in some situations.

The other option, adding a constant parameter identifying the specific method, IMHO, is more clutter. The algorithm is fundamentally different, and different methods to attain similar results may require additional parameters (ex, a max iterations parameter might make sense for one approach but not another). You also end up with either an enumeration specific to that one method, or an inability for the compiler to check the validity of the enum being passed in (ie, you have implemented NR and ISR, but not BSR; what happens when the BSR constant is passed in?) The enumeration and the code implementation get too far separated in all but the simplest of algorithms, and so will get out of sync.

Jul '08

Syd

I used to believe that well-written code was self-documenting... then I started working with a lot of junior devs!!

My philosophy now is to comment almost EVERY line, explaining the business rules - I find it encourages the junior devs to think about what they are doing. (And if you don't enforce comment every line then (because developers are all lazy) then the comments drop down to zero very rapidly and you end up with no comments at all.)

For example:

```
'Get all the customers who live in this state

dim objCustomers As New Customer = Customer.GetWhere(State = 'VA')

'If there are no customers in this state...

if objCustomers.Count = 0 Then

    '... then we warn the user

    MessageBox.Show(You are ALONE!)

End If
```

Yes, the comments are strictly unnecessary but they may help someone's understanding at some point in the future, and they certainly help mine when I am code reviewing, so... they stay!!!

Jul '08

Ben

(Has Jeff jumped the shark?)

If you're saying that comments are useless if they are no clearer than the code, that's not exactly a revelation.

If you're saying that well-written code is always clearer than a comment, that's bo11ocks.

Comments are as much note to self as to anyone else:

1. I'm maintaining some code, it references some class. The code for the class is three pages long, but I just want to know what the class is there for. Shame that the guy who wrote the class (was it me?) considered that the code was self-explanatory.

2. I have a constant K_WIDTH=100. Cool, the coder was clever enough not to name it K_W. But why not 200?

3. Pushed for time, I haven't had time to refactor as I would have liked. How about a little //TODO? (Or //HACK?)

4. I'm maintaining some code, there is a 10-line procedure which could be replaced with just one method call. I think. Did the guy who wrote the code think of it? Is there some good reason to use 10 lines that I haven't seen?

5. (In C#) If I prefix the comment for my class or method with ///, hey, any client code gets extra information in intellisense. (That's even more useful if all they've got is the dll and no 'self-evident' code!)

6. I have a for loop with 5 method calls inside. Do I have to go and look at the code for the 5 methods to work out why the loop starts at 1 instead of 0?

OK, so all my examples are probably more why than how. My point is that you can ignore comments that *are* there, but you can't read comments that *aren't* there. Please don't give any extra ammunition to the 'write-once, read-never' cowboys coders who are just too lazy to make the effort.

How about you give us some examples of good *and* bad comments?

Jul '08

Sam_Hill

I read english a hell of a lot quicker than I do code, give me a well written comment any day. And sure, give the method, variable whatever a descriptive name but don't skip the title. CamelCaseIsn'tThatGreatToRead.

Jul '08

Syd

PS - And, yes, I still enforce Hungrarian (sic!!) notation - otherwise (especially in case-insensitive VB) you end up with variables called the same thing as your classes, which ends up VERY confusing, especially if you use static methods a fair bit.

Jul '08

Baz_L

Ok, here I go disagreeing with you guys.

These points reflect what would be done in an ideal world. And 2 years ago, I would agree with everything said.

HOWEVER, after working on a Monster of a project for the past two years, I've began yearning for comments in code. A lot of my work deals with bug fixes in existing code and I've finally quelled my urges to simply rewrite crap that doesn't work. Here are my reasons:

- Sometimes past developers were just simply bad and there's no 'time' to refactor.
- Significantly modifying (and trust me, a lot of the modifications needed would be significant) would require retesting that functionality in the QA department. In an environment where time = money, that just isn't a viable option.
- Comments are usually frowned upon where I am (They get in the way, I've been told). So the alternative is to sit (sometimes for hours) debugging trying to figure out what's going on.
- In my environment, the 'weird' parts of the code usually deal more with business logic and less with actually functionality. When there's a statement that says: {if A = 'CPTY'}; I'm sorry, I need a comment. Because walking over to the support department to ask what's CPTY isn't my idea of the best use of my time.

So, what does one do with messy code when refactoring isn't an option? I say, comment the hell out of it.

Jul '08

John_Grimes

You've incorrectly asserted that too many comments are bad, as evidenced by a large chunk of useless comments. Useless comments that add nothing to the code are just that, useless. But for complex methods that take several individual tasks, I like to start out by commenting what those tasks are. When I revisit my code, I can simply read the comments to see a breakdown of each step and what I was thinking at the time, without having to ever look at actual code.

Sure, a simply for loop extracting a value from a bean and placing it in a hashtable is pretty simple to understand if you're not uses cryptic variables, but I just can't see that
//loop through bean and place value in hashtable
isn't still adding value.

Jul '08

Shoban

I once saw a comment confused in one of tha web applications I support!!

Took some time to realise that this guy who wrote the code years back was refering to XML messages sent to confused.com

Jul '08

Gareth

 alex13:

```
r = n / 2;
```

```
while ( abs( r - (n/r) )  t ) {

  r = 0.5 * ( r + (n/r) );

}

System.out.println( r =  + r );
```

Isn't (n/r) always 2?

There's a while loop there. In successive iterations r will be something other than n / 2 so n/r won't be 2 any more

Jul '08

James

- Tom Dibble

Yeah the Newton-Raphson method is generally called just the Newton method. I agree in some cases it's best to append a comment explaining a little bit of what's going on. But what I meant is also something you proved yourself. If I see SquareRootApprox(n) and I know that Newtons method is the most common for approximation of square roots, then I don't need to look any further into it. But if I see SquareRootApprox(n) or // Newton-Raphson Square Root Approximation, and I don't know what it is or the specifics of the algorithm and I type either in google... I'm gonna end up reading the same thing. The only other possibility is that the programmer inserts a comment as long as a blog post explaining how the Newton method works... and that just makes everything bloated and hard to maintain.

My issue with comments is that too many comments makes everything uncomfortable and a hell of a lot harder to maintain. I've been in situations where I have a file maximized on my 30 inch monitor and I'm looking at huge blocks of comments and 2 or 3 lines of code. I ended up removing the superfluous comments and rewriting a big part of the code. Nevertheless, when we got new developers on board, everyone got up to speed fairly quickly and everyone praised the naming conventions and wise use of comments in the code (which where probably less than 1% of the total lines of comments previously there).

Jul '08

ToddM

Not sure if you inspired this, or if it's just a coincidence but it's a comic that seems kinda relevant to the discussion: http://geekandpoke.typepad.com/geekandpoke/2008/07/good-comments.html

Jul '08

Atario

I pretty much only use comments in two circumstances:

1. When what I'm doing in the code seems to make no (or not enough) sense. These tend to be either things like If you don't do this way up here, you get an Off-By-One error or else large blocks explaining that some object's implementation is brain-dead, must be worked around, and ending with Thanks a lot, [object vendor]. Usually [object vendor] = Microsoft, but Crystal Reports has been a suspect in the past.

2. XML comments intended for Intellisense consumption. This is a recent development for me and I'm still wary about it.

Jul '08

Jin

comments are merely words.

Jul '08

PaulW

It’s easy to make the same mistake with your user interface.

“Oh, this interface is a bit confusing. Let’s put some explanatory text in so that users understand what’s going on.” FAIL.

Jul '08

Trevor

Pi is exactly three!

Jul '08

roflmao

@Atario - 'XML comments intended for Intellisense consumption. This is a recent development for me and I'm still wary about it.'

Are you taking the piss mate, or were you just asleep for the last 7 years?

Jul '08

Onur

Yeah. I've already talked about it: (but in french)
http://sexycoders.com/index.php/2008-03-13/pourquoi-ne-pas-commenter-son-code/

Jul '08

pedro1

I think that comments are generally a good thing, as someone said about you can ignore the ones that are there but can't read the ones that aren't.

On my daily job I usually have tight deadlines that become tighter near the end, and that makes the code quality start to decay plus I have juniors on my team, so comments are a must. For example I just wrote:

```
//find cases in where this solution is relevant

//note: Last minute change, may be changed to backend at a later date.

protected function inferRelatedCases(solution:SolutionVo):String
```

This is not only a note to myself but also for someone in the future that comes across my code and wonders why this is here.

Jul '08

Greg_Beech

Weirdly, your post started out making a very valid point (tell us why the code is doing what it is, not what it is doing) but then you proceeded to argue an entirely different point about why you should refactor your code to make it is more clear what it is doing.

So great, the new refactored code makes it clear that you're doing a Newton-Raphson approximation to the square root. But I still don't know WHY you're calculating a square root, or WHY you chose the N-R method (efficiency, accuracy, something else?).

The important comment here would be something like:

```
// using the N-R algorithm as it typically converges quickly // to
approximately the desired root. we don't need the exact // value here
because {reason} so this gives us better perf.
`
```

Jul '08

Greg_Beech

Sorry for the double post  but here's an example of what I mean. Here's a method that removes a subscription in a catch block, taken from our real codebase. It is very obvious as to what it is doing, but do you know why?

this.Unsubscribe(subscription.Id);

Now does this comment help?

```
// if the operation was invalid when we know the cause must be an invalid
workflow queue and/or the workflow

// not existing because the only other cause from the methods in the try
block is that the workflow runtime

// is not started, and that can't be the case here as we just checked. as
we'll never be able to deliver the

// result to a non-existent queue/workflow then we need to remove the
subscription otherwise it will stay in

// the pending work queue for runtime services that persist their jobs and
cause the service to enter an

// infinite loop processing work items that it can never complete

this.Unsubscribe(subscription.Id);
```

Jul '08

GunstarC

As far as I understand the re-written code, it doesn't work.

't' is never declared.

Unless it's a global? And I know someone as precious as this about a simple thing like comments wouldn't go around advocating a global.

Jul '08

pete

I find I put one or two comments in as I am working on a couple of methods then when I refactor sometimes they become unnecessary. Once I have got to a stopping point like written a set of methods or a full class I will have a quick scan through adding in some more general comments about what a method does if it isn't apparent in the methods name.

Don't comment with pseudo code most programmers can read the actual code easily. A comment should be a sentence covering what a set of lines do so you can skip them when debugging if you know they work.

Jul '08

Mark

My first thought on reading the post title was: comments are not to be messed with.

But after reading your and Sammy Larbi's posts I took a look at my more recent code... and realized you guys are right. I still think complex code needs comments, but as you choose descriptive names for your methods the code should be self explanatory.

Posted an example of my own code on my blog: http://www.markvandenbergh.com/archives/21/no-more-comments-in-your-code/

Jul '08

Nick_Waters

I've heard this before, and I consider it disingenuous. Only a relative few are able to read code as effectively as a code interpreter. I've worked with many developers over the last 20 years and I can't think of a single individual who could do it perfectly.

What does this post really mean? Should everyone unable to read code as efficiently as the author go dig ditches for a living? The vast majority of coding is done by average developers who may need comments to help them along the way to comprehension.

This is especially true of maintenance development where coders are reading code which may be many years old and have numerous coding styles, with little or no documentation to follow as to the purpose of every function or feature.

Jul '08

Dan

My response to Jeff's post:

http://blog.uncommons.org/2008/07/25/no-your-code-is-not-so-great-that-it-doesnt-need-comments/

Jul '08

xxx

I haven't added a single comment, and yet this mysterious bit of code
is now perfectly understandable.

Actually, no. I *still* have no idea what the hell that algorithm is and why it works. OTOH, // square root of n with Newton-Raphson approximation tells me all I need to know in order to be able to find more information about the algorithm, should I need it.

Jul '08

Kevin_Dente

Agree 100%. If you can't tell what the code does without comments, the code isn't done yet.

Jul '08

Rob

Whats a comment? Whats documenting?

My 10 bits.

Comment as a prototype of the code, just to get my idea of the code out.

Replace the comments (not underneath) with the code, often being a meaningful method name.

Add comments to highlight any quirks, notes, methods used, reasoning why. e.g. Uses X method to do some awesome sh!t to the string, This code is sluggish, but I cant think of a better way, tried x,y,z.

Jul '08

Bill79

It's funny how often I see people complaining about out of date comments then not fixing them.

Most of the comments make the code less readable argument is simply looking for an excuse not to have to explain what their own crappy code does when they often don't understand it themselves.

Personally I have really wanted comments quite a few times just to judge the intelligence and intentions of the author. Did he know what he was doing when he added this apparently extraneous variable? Often you don't find out that it was actually a key hack added at the end to support some business rule until you're in the middle of removing it.

Any time you hit code that you don't get within a few seconds, be it yours or someone elses, FIX IT. Either add comments (which has the benefit of virtually never breaking the code) or refactor the hell out of it until you have something understandable.

If you just leave it and work around it, letting the next guy start from scratch as well, you're worse than the original author because you KNEW there was an issue.

Basically your code should read like a book when you are revisiting it (not when you are writing it and it's fresh on your mind). There should never be a question as to why you are doing something.

As you pointed out, this can often be done by very small single purpose methods and classes that are appropriately named (Anyone that thinks that simply naming things well will do it is part of the problem though).

Honestly it's a LOT more work to make readable code than to write a few readable comments, and I doubt a lot of programmers are even capable of writing readable code--then someone comes along and gives them the idea that using a 30 character method name for a 200 line method is going to absolve them of all responsibility.

Jul '08

Jax

Comments are for anything unusual in my books.
Anything that isn't immediately obvious or would look a bit wrong without a comment. This means that they describe either some form of hardware or architectural limitation, a fixed bug, some performance hi-jinks or a curious customer requirement.

Oh I also use them to explain to readers that i've considered certain code paths and am happy they do nothing.

```
if(blah)

{

blahblah

}

else if(blah)

{

blahblah

}

else

{

// do nothing

}
```

Jul '08

Mecki

Well, I partly agree... and then I disagree again. First of all

COMMENTS ARE GOOD!

Never think otherwise. Trying to avoid comments is a step into the wrong direction. It's not about avoiding comments. It's about getting your comments RIGHT(tm).

As other people pointed out, by choosing the right name for the function, you made clear, that this function calculates a square root. But you don't tell people which method it uses and why you chose this method.

This gets me to the point:

1) Don't use comments to tell people what you are doing, unless it is very hard to see that by just looking at the code. Always assume that whoever will read your code is a good programmer and knows the language your code is written in quite well.

Most useless comment I have ever seen:

```
// Increase i by one

i++;
```

<sarcasm>Wow, thanks for the comment. Without it I had never guessed what happens there! </sarcasm>

If somebody needs a comment to know what i++ does, he shouldn't be reading this code in the first place.

But your square root example is a good example where I would leave the comment, even if you chose the function name to be more obvious, the name does not reflect which method you use for calculating the square root. I had at least written that as

```
private double SquareRootApproximation(n) {

// Using Newton-Raphson approximation

...
```

2) Always use comments to tell people WHY you are doing something.

The what might be obvious by just looking at the code, the why certainly never is. How can it be? The why exists only in your head. It's a design decision you made while writing the code and maybe not even you yourself will know 3 years later why you made it that way and not any other way. So

make sure the why is either always obvious (because it's trivial) or add a comment that explains your design decision.

3) Don't over-comment.

I guess that was your main critics here and I totally agree with you. A lot of code has more comments than actually code. If that happens, something is wrong. It's highly unlikely that 100 lines of code are so complicated, that they need 300 lines of comment to understand them.

If you really implemented something so ingenious, that you want to share your cleverness with the whole world, write a 50 pages PDF file, upload it somewhere on the web, add a two line comment to your code that contains a link to it. Who really cares for a long winded explanation of what this code does, why it is so cool, a proof that it really works, a performance comparison, and so on, can grab that PDF and read it. For the rest of us, just add a comment what the code does, why you decided to do it that way and that it is really great - we will believe you 

Jul '08

Alec_Flett

While I absolutely agree about properly factored code and good naming conventions, I think this conclusion is naive:

if your feel your code is too complex to understand without
comments, your code is probably just bad. Rewrite it until it
doesn't need comments any more. If, at the end of that effort, you
still feel comments are necessary, then by all means, add comments.
Carefully.

It only works in simple systems where you, the singular developer, control the entire codebase, any you're not interacting with any other systems. For instance, it works great for pure data-in/data-out algorithms.

Reasons for commenting usually have little to do with the quality of the code itself, but rather a limitation of the architecture you're living in - for instance, the web.

When your application is spread across server and client code, UI code/resource bundles, CSS, etc, and depends on the quirks of HTTP caching, SQL performance characteristics, third party libraries, etc, you absolutely need comments to explain yourself.

Jul '08

MarkM

Whilst I don't like essays as comments I still think adding that one line tidbit you'd added is useful.

```
// square root of n with Newton-Raphson approximation
```

Don't get me wrong, I'd still probably stick it as some sort of static method in a utility class somewhere but I like to know a little about the logic being used, and I think it becomes more relevant with more complex examples.

As an example this week I was working on some code to calculate the distance between spherical coordinates (lat, long), for my purposes I decide that the haversine formula - with a rough error of 3m for every km - was suitable. To me putting in something like

```
// haversine formula - error margin 3m per km
```

makes sense. Any other developer would be able to see it's limitations and determine if this was suitable for their needs.

Jul '08

SB11

Holy crap, is this bizarro world? Programmers are writing TOO many comments? I've never heard such a thing! I wish this was in line with my experience.

Just because I am a programmer and I can read code, does not mean I should have to. I'm not a compiler. Sure, I completely, 100% agree that code should be self-documenting where ever possible. On the outskirts of code, you will likely have some type of controlling logic which creates objects and calls functions, all of which should be fairly self-documenting and the flow is obvious and beautiful. These are the trivial cases. But as you get deeper in the meat and potatoes of the code, it becomes harder and harder to understand the code as immediately as the trivial parts. Instead of reading the code like a book, you have to start parsing the code like a compiler. It's inevitable. Not everything can be broken into a 4-line function with a nice name. In these cases, a simple // XX does YY and then ZZ makes all of the difference in readability. Not only that, but it shows the programmer's intent. If there happens to be a bug in this area, I can identify what the programmer was *trying* to do, why her assumptions were wrong, and thus fix the bug. Self-documenting code is completely worthless if it's inadvertently documenting the wrong thing.

Really, the issue here, is the maintenance phase of the programming life cycle. As we know (From Facts and Fallacies), the maintenance phase consumes 40-80% of the total costs of a project. And maintenance programmers spend 50-60% of their time deciphering the code they are maintaining (Code Complete, referencing another source). So the question is, do comments help this deciphering phase? Yes. They do. They help more than not having them, that's for sure! I'd rather have too many comments than not enough. I think it's plainly obvious, but I'm sure there are studies that support it. Here is one with a few seconds of google searching:

http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/trans/ts/toc=comp/trans/ts/1988/09/e9toc.xmlDOI=10.1109/32.6171

Any time you have code that is going to need published documentation, JavaDoc-esque systems do take a lot of space comment-wise, but if you need to maintain documentation anyways, it's easier to do it right there in the code. It just takes a bit of discipline to remember to update the doc block if necessary whenever you are doing something in the code. Much more convenient than maintaining documentation elsewhere. And it may be useful to other programmers looking at the code, too.

It seems like there are a lot of posts on this blog and associated blogs with the tone of: Generally accepted good thing here is actually TERRIBLE!!! OMG!!!. This is reminiscent of cable news network scare tactics and nonsense to start controversy when there really shouldn't be any. Yes, using comments, design patterns, regular expressions, XML, etc to the *extreme* is bad. Of course. This could be said about anything. But what should be known is that they are generally *good things*, but should be used correctly, with care, and under the appropriate circumstances. Self documenting code and comments are both things a programmer should use and continually attempt to get better at.

Preaching the correct way to use these things is probably going to be much more helpful for those you are trying to address. I see that you aren't saying all comments are bad- But that is certainly the

tone. It's not until the last sentence that you admit there may be a use for them. I think people like you, Sammy, and others currently on an anti-comment binge are only going to hurt yourselves if you are so strict and unwavering in regards to code commenting.

The effort it takes to write them is *vastly* (and I mean VASTLY) overstated and the space it takes up on your screen an irrelevancy, as far as I'm concerned (Ignore the dark-green text!). You act like a programmer needs to be a novelist to write simple comments. I think I've seen people have a harder time naming functions appropriately.

In short, I don't think you are young enough to be so idealistic 

Jul '08

PaulH

It's not possible to overcomment something in reality.
This is another one of those bull$hit macho, 'I don't need comments in MY code' threads that should become ignored.

Jeff, why don't you go and re-read one of your favourite books, the Mythical Man Month, to hear all about this discussion and why you are wrong?

Jul '08

slapout

Not quite the same thing, unless those comments stay in the code after you've written it!

But, that's what Steve Mcconnell suggests in Code Complete.

Jul '08

FrancisF

Another related issue is use your source code control and don't use comments as a kind of historical record of code that has been replaced. It works for a couple of iterations in case your refactoring broke something - maybe - but 20 iterations later it's plain dangerous.

Be brave, delete stuff and trust in your source code control system.

Jul '08

matt16

But I like telling myself stories as I wrote code! Honestly, the layman's-terms expression of the function of whatever I happen to be developing is actually helpful, I've found, in keeping my own thoughts straight. So while I might have 4 times as many comments as you might recommend, I feel like my development process has been smoother and less convoluted. It probably helps solidify new concepts too.

However, before I hand my code over to another developer, or particularly before I deliver it, I strip out all my 'note to self' comments and rewrite things a lot more sparsely. It's kind of redundant, I know, but it works for me, so I figured it'd be worth mentioning.

Jul '08

J1121

What about when you're in a real world situation and you've just got to make a deadline? You can't begin to refactor your code for ten times, you've just got to get that shit out of the door.

Jul '08

Steven_Bey

```
bool comment = false; // indicates whether this comment is required
```

Jul '08

JonathanS

Jeff,

If you are writing code that be consumed internally (not for public consumption), then comments can be minimal especially when you use excellent naming conventions for classes.

But when you write code others will use, it's really nice to provide many more details with your comments. I find the example and code tags to be really nice to include in documentation I have written code for others to consume. Even though I use excellent naming conventions as well.

Jul '08

Dave

Reminds me of a GREAT article I saw a while ago about (funny) tips for creating unmaintainable code:

1. Choose variable names that masquerade as mathematical operators:
   openParen = (slash + asterix) / equals;

2. Choose variable names with irrelevant emotional connotation:
   marypoppins = (superman + starship) / god;

http://mindprod.com/jgloss/unmainnaming.html

It's really the opposite of self documenting code, it's more like self-obfuscating code. 

Jul '08

Aaronrs

Self-documenting code is completely worthless if it's inadvertently documenting the wrong thing

Self-documenting is by definition the right thing. It may be poor code and wrong, but it is the correct documentation.
Comments rapidly become out of date with refactoring, and with fellow programmers not updating it.

```
// Newton Raphson method

private double SquareRootApproximation(n) {

  (some other algorithm)

  ...

}
```

Writing tests before code documents the requirements by defining the inputs and outputs. No code comments required.

Also, why are you reading the code if you don't know why or how you got there? Junping into the middle of a program without an aim is pretty pointless.

Jul '08

James

this post reminds me of the NeHe OpenGL tutorials, where every single line, including opening and closing braces are commented.

Comments in my code are just that - comments; little side notes, kind of what you'd note down next to someone else's code if you were tring to work out what it did.

And if you hate giant comment blocks, don't view source that's been formatted for Doxygen.

Jul '08

f0dder

If you're doing Newton-Raphson, I hope you're doing so for a reason other than demonstrating your Mighty Math Muscle. State that you're using this algorithm, and why. State the accuracy bounds, and enforce via tests. Don't leak implementation detail in the interface.

Jeff's SquareRootApproximation(n, Type.NewtonRaphson) examle is bad for reasons already mentioned, it's clunky, and depending on language+compiler might generate suboptimal code. If you're doing N-R it's usually because of speed concerns, so you don't want suboptimal.

I do fully agree you shouldn't comment every single line, though, and maintenance coders not updating comments *can* be a problem. But would you want a maintenance coder on your team who changes algorithm without updating comments? Chances are that coder won't be writing proper tests, either.

As for Junping into the middle of a program without an aim is pretty pointless. - debugging other people's code, perhaps? 

Jul '08

Leo3

It's reassuring to read the replies and find that so many people don't buy the less comments are good nonsense.
There are two main reasons that lead to this incorrect thinking:
1. Developers think that code tells you what is being done and comments should tell you why it's being done. This is nonsense. Code tells you how something is being done, and comments should tell you what is being done on a semantically richer level than instructions intended for a machine. Even better if they also tell you the why.
2. Developers regularly overestimate the self-explanatory power of code they have written themselves. It's only experience that teaches them to take nothing for granted. Please don't tell inexperienced programmers that comments should be avoided.
I agree, however, that writing good comments requires good writing skills that many coders do not possess. Again, it doesn't help to reassure these that comment-free code is good, because without practice, they never gain the necessary skills. In my opinion, comments and code intimately belong together and should be improved together.
By the way, I think that programming skills correspond with writing skills on a very general level. Bad writers confuse ideas, present them incoherently, get character traits wrong, use inconsistent naming, grammar and punctuation, and tell their stories in an unimaginative, boring and sloppy way, not detecting mistakes and limiting story development through these errors. Bad coders confuse and misunderstand requirements, structure code incoherently, get abstraction levels and data structures wrong, use inconsistent naming, idioms and punctuation, and sloppily implement algorithms that are unimaginative, inefficient and buggy, limiting maintenance and enhancement possibilities. I believe that good programmers are mostly good writers and vice versa. As a corollary: Most software is buggy, inefficient and hard to maintain because their authors suck at writing.
You could go a long way in telling people how to write and code correctly, Jeff, and improve their coding abilities. For my part I'd look forward to again read an inspiring post from you. It's about time.

1 reply

Jul '08

MikeW

Apprentice: either writes comments, lots of comments or writes incomprehensible code with no comments.

Journeyman: writes self-documenting code with no comments.

Master: Writes comprehensible code that is commented where doing so enhances understanding.

Something like that?

1 reply

Jul '08

codinghorror

L Leo3:

For my part I'd look forward to again read an inspiring post from you.

Do I really need to write a post saying comments are good, please use them? That's like writing code is good, please write it.

http://www.codinghorror.com/blog/archives/000878.html

If the best code is no code, what is the best comment?

M MikeW:

Apprentice: either writes comments, lots of comments or writes incomprehensible code with no comments.

Journeyman: writes self-documenting code with no comments.

Master: Writes comprehensible code that is commented where doing so enhances understanding.

+1

Jul '08

J

John_Pirie

Couldn't agree more about the thrust: comments should be used sparely, and code should document when at all possible.

But let's face it -- anyone replying to this post is part of the 5% of programmers who actually give some thought to the topic while working.

It's the other 95% that's the problem.

Jul '08

Henning

What the hell???

Instead of adding a single line of comment, you recommend writing the stuff as a function/method, and comment it by giving this function a speaking name - even though the code is onl run once, and no function is necessary(otherwise it would be there anyway)?!

Sorry, that's not real affective, and will, for all those morons that make other developer's day hard by not being able to give functions a decent name, not make anything better.

And then the You should always write your code as if comments didn't exist..

Totally mislead - as when you're walking along the equator to find th north pole (not as wrong a s on the south pole, but here, not even the temperature is right...)

I code by writing the comments first - saying in human language, what I'm actually about to do, and, by that way, realizing problems in things because I have problems to describe them - so, before the first line of actual code is written, I realize problems in my spec(if such thing ever exists - in real life you don't get these so easily).

When I write unit tests, the first thing I do is try to write prerequisites and expectations into the comment above the code - before starting to write the test. As with the coding itself, it makes me reassure, that I have enough inside in the problem to be solved so I can describe it in a short sentence - if I'm lacking this, I will never be able to write a decent line of code doing these things.

Even though I don't want to force that style of working to others, I feel it's working great for me - it leads to well thought out coding, and can *never* lead to undocumented code.
However others work - what you describe and recommend here is nothing else to me but the title of this website...

Jul '08

tragomaskhalos

The solution, of course, is to use the Commentator:
http://www.cenqua.com/commentator/

Jul '08

Henning

Oops, wrong URL - get me here: http://lazyb0y.blogspot.com

Jul '08

Vincent

I guess we all tend to fall into the overstated tone of Jeff's articles, but like those of Joel, that's what makes them so enjoyable and we just can't help reacting.

I think the overuse of comments if one of the 'code smells' described in Martin Fawler's Refactoring: Improving the Design of Existing Code, and a more reasonable opinion would be at least to try to improve the code until most comments become useless. I personally spend a ridiculous amount of time trying to remove all boilerplate code and extracting every bit of logical blocks to new methods with explicit names, so I totally buy that coding without comment style, but only because I'm way too lazy to keep comments up-to-date, plus I think it pollutes my editor screen. But yeah, I feel guilty about that.

Jul '08

silence

I suppose the level of detail in comments depends on the level of understanding by the people looking at the code; where I work, not all 'programmer' are at the same level (I wouldn't even consider some of them programmers at all) so whenever I loop through an array I have to write down what's happening otherwise I get people ringin me up asking me what's going on.

Jul '08

BrianG

Jeff,

I couldn't agree with you more. In fact, I wrote a very similar post 7 days ago about the exact same thing (shameless plug: http://houseofbilz.com/2008/06/comments.html). I think my language was a bit stronger against comments, but your language was clearly more articulate.

Jul '08

Leo4

Jeff,

Do I really need to write a post saying comments are good, please use them? That's like writing code is good, please write it.

No. Make people reflect. Let them benefit from your experience. Show them the how, not the what, and let them draw the conclusions on their own. Put things into perspective, rather than advocating one side. You've got an audience, so you've got responsibility.

If the best code is no code, what is the best comment?

Well, if the best code is no code, then the best job is no job. Ex falso quodlibet

Sure, I agree that less is (sometimes) more. But you also know that there's a limit to quality by reduction. Why shouldn't the same be true for comments?

Jul '08

structure

you are a shameless flame-baiter

Jul '08

DaveA

Another excellent compromise that you skipped, is meaningful variable names! I am not familiar with Newton's Method (might have seen it in school, *coughcough* years ago), but it looks like r is the root and t is a tolerance. Why not call them that? while (abs(root - (n/root)) tolerance) would have been clear enough that I could figure out at a glance what that was trying to achieve.

Jul '08

DaveA

Harrumph. I *did* check to see if anyone had already commented on the var names, but for some reason the web site only showed me the comments written up to Dan's of 08:09 PM last night. Methinks there's a naughty cacher somewhere on my slow link. Sorry! B-(