Clean Code

by Robert Martin

Presented by
Nicholas Dwork
5/8/15

What is Clean Code?

I like my code to be elegant and efficient. The logic should be straightforward to make it hard for bugs to hide, the dependencies minimal to ease maintenance, error handling complete according to an articulated strategy, and performance close to optimal so as not to tempt people to make the code messy with unprincipled optimizations. *Clean code does one thing well.*

- Bjarne Stroustrup
(Inventor of C++)

What is Clean Code?

You know you are working on clean code when each routine you read turns out to be pretty much what you expected. You can call it beautiful code when the code also makes it look like the language was made for the problem.

- Ward Cunningham
(Inventor of Wiki)

What is a Code?

It's what we do.

Cost of Dirty Code

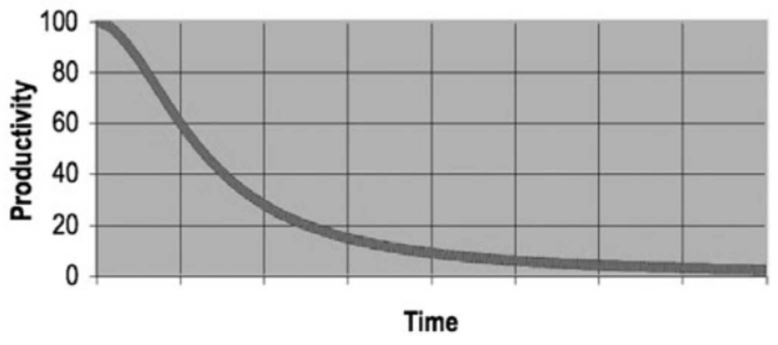
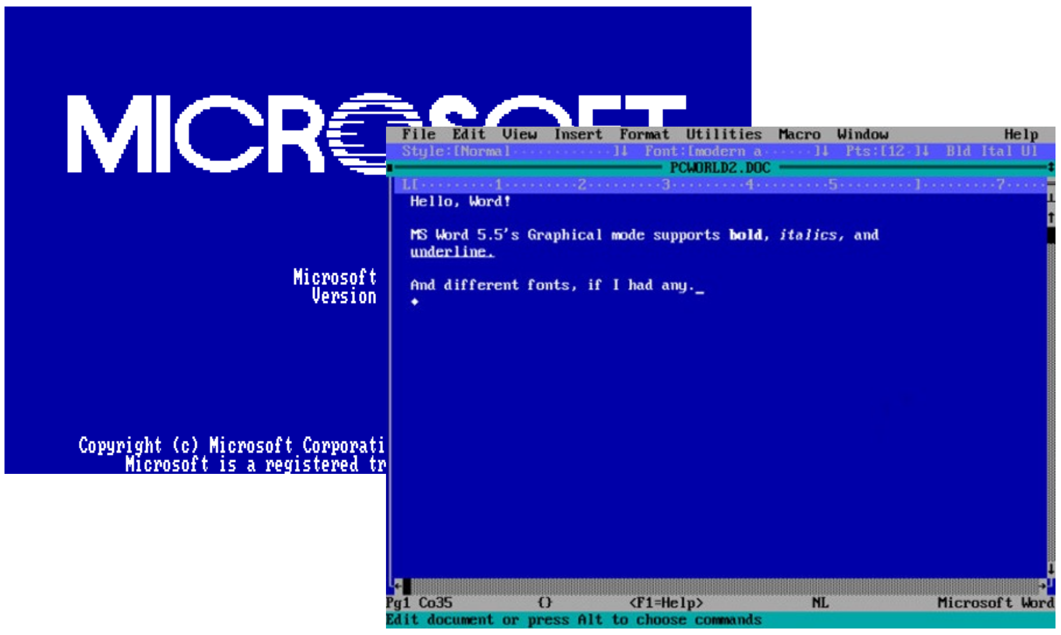


Figure 1-1
Productivity vs. time

Microsoft Word 1.0



MICROSOFT
File Edit View Insert Format Utilities Macro Window Help
Style:[Normal] Font:[modern a] Pts:[12] Bld Ital Ul
PCWORLD2.DOC
Pg1 Co35 {} <F1=Help> NL Microsoft Word

Microsoft
Version

Copyright (c) Microsoft Corporati
Microsoft is a registered tr

Hello, Word!

MS Word 5.5's Graphical mode supports **bold**, *italics*, and underline.

And different fonts, if I had any._

Edit document or press Alt to choose commands

Refactor as you Go

Leave the code cleaner than you found it.

“Technical Debt” (term used at Google):

- Deadlines, budgets, and experimentation lead to dirty code.
- The code becomes increasingly unwieldy: difficult to read and understand, hard to debug, difficult to add features
- Pay off the technical debt: refactor the code, make it easier to read and understand

Variable Names

Make variable names nouns.
The name of the variable tells you why it exists.

d = 8;	Not descriptive.
elapsedTime_days = 8;	Very clear.
kx = 8;	Good
kx_cyclesPerCm;	Could be better

Avoid Magic Numbers

Use variables rather than hard code numbers.

```
area = 0.5 * 3 * 4;
```

Unclear what's happening.

```
base = 3;
height = 4;
area = 0.5 * base * height;
```

Much more clear that we're calculating the area of a triangle.

Functions

Make functions perform exactly one task.

Make functions short.

Rule of thumb: 20-30 lines

Function Names

Function names start with verbs.
The name of the function tells you what it does.

d = ld(name)	variable names aren't descriptive not clear what ld does
data = loadData(filename)	The intent of the function is clearly understood

Make Functions Accept few Arguments

The ideal number of arguments for a function is 0.
The next best is 1.

The fewer the number of arguments, the easier the function is to understand and test.
  Think about the combinatorics.

Argument names should be descriptive.
  Same rules as variable names.

Have Functions Maintain a Level of Detail

```
function playTone( )
    toneData = loadTone( toneFile );
    toneData(1:100) = 0;
    sendToneToSpeaker( toneData );
end
```

Highlighted line is weird. It's at a much different level of detail than the rest of the function.

```
function img = reconstructMriImage( pfile )

    kData = loadData( pfile );

    griddedData = grid( kData );

    img = reconMRI( griddedData );

end
```

By keeping the function short, and modularizing appropriately, dependencies are made clear.
- it's clear that reconstruction requires only the gridded data, and not the k-space data

Avoid Boolean Flags

Boolean flags are a sign that your code does more than one thing.

Test Driven Development

Write the unit test before you write production code

Write just enough production code to pass the test

With TDD, code development is a cycle.
Test and production code are written together.
Tests are just one step beyond production code.

Unit Tests

Make unit tests independent
Don’t let tests depend on each other

Make unit tests fast

Make sure all unit tests pass before committing code

Have a script that runs through all tests

Make test results clear:
FAILURE FAILURE FAILURE

```
function testE_adjointE
   x = rand(10,1);  y = rand(100,1);
   innerProd1 = sum( applyE(x) .* y );
   innerProd2 = sum( x .* applyAdjointE(y) );
   diff = abs( innerProd1 - innerProd2 );
   disp(‘Testing <Ex,y> = <x,E^Ty>:’);
   disp([‘    Should be 0: ‘, num2str(diff)]);
end
```

Test is ok
Doesn’t guarantee applyE or applyAdjointE work
Would be much better if it output Passed or Failure

Test code is just as important as production code.

Test everything you can.

Comments

A Necessary Evil

- It is very easy for comments to be out-of-date, misleading, or wrong since they are not enforced by the compiler.
- Only use comments when absolutely necessary.
- Comments should be avoided; information should be conveyed through descriptive names.

Avoid Duplication

Duplication is a sign that the code isn’t modularized properly.

Copy and pasting is a cardinal sin

Successive Refinement

Make code get better with age.

5 More Bugs

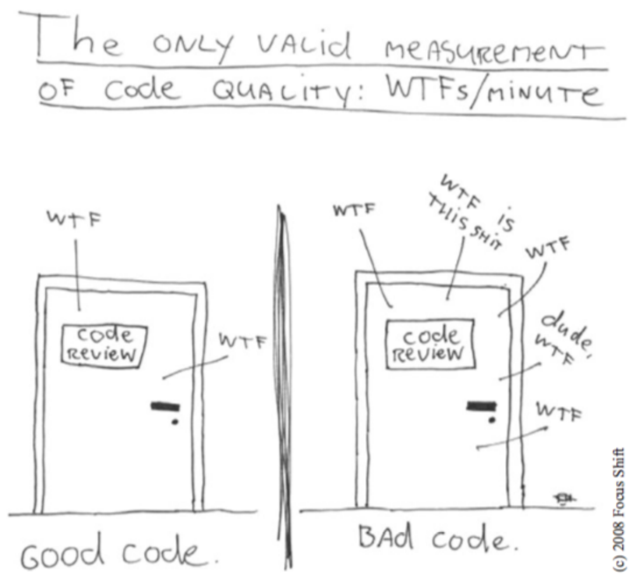
There are always 5 more bugs

The biggest hurdle to overcome in debugging is denial that there is a bug.

-Brian Hargreaves

Conclusions

- Descriptive
- Small
- Does one thing
- Tested


The only valid measurement of code quality: WTFs/minute
WTF
code review
WTF
Good code.
WTF
WTF is this shit
WTF
code review
dude, WTF
WTF
Bad code.
(c) 2008 Focus Shift