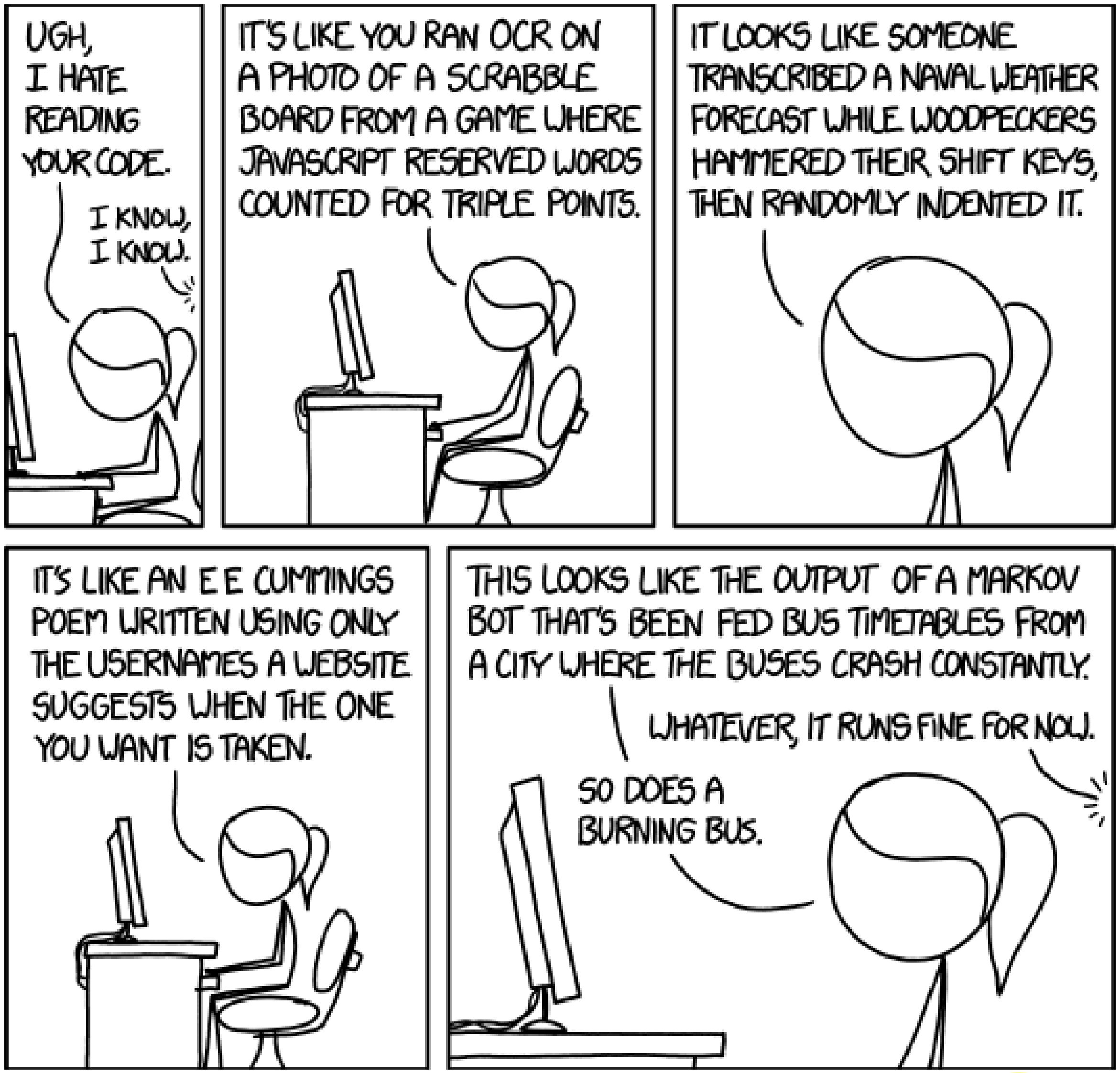
WRITING CLEAN CODE

By Pablo Restrepo

Based on the book "Clean Code" by Uncle Bob

FORMATTING

Bad Code


UGH, I HATE READING YOUR CODE.
I KNOW, I KNOW.
IT'S LIKE YOU RAN OCR ON A PHOTO OF A SCRABBLE BOARD FROM A GAME WHERE JAVASCRIPT RESERVED WORDS COUNTED FOR TRIPLE POINTS.
IT LOOKS LIKE SOMEONE TRANSCRIBED A NAVAL WEATHER FORECAST WHILE WOODPECKERS HAMMERED THEIR SHIFT KEYS, THEN RANDOMLY INDENTED IT.
IT'S LIKE AN E E CUMMINGS POEM WRITTEN USING ONLY THE USERNAMES A WEBSITE SUGGESTS WHEN THE ONE YOU WANT IS TAKEN.
THIS LOOKS LIKE THE OUTPUT OF A MARKOV BOT THAT'S BEEN FED BUS TIMETABLES FROM A CITY WHERE THE BUSES CRASH CONSTANTLY.
WHATEVER, IT RUNS FINE FOR NOW.
SO DOES A BURNING BUS.

Formatting

If you are working on a team, then the team should agree to a single set of formatting rules and all members should comply.

It helps to have an automated tool that can apply those formatting rules for you.

Purpose of formatting

Formatting is important.

He have seen throughout this course that code readability is VERY important. when your code changes your discipline and style usually survives.

The readability of your code will have a profound effect on all the changes that will ever be made

Vertical formatting

It is possible to build significant systems out of files that are typically 200 lines long, with an
upper limit of 500.

Although this should not be a hard and fast rule, it should be considered
very desirable.

Small files are usually easier to understand than large files are.

The Newspaper Metaphor

Your code should read like a newspaper.

Usually in a newspaper you will find a headline that tells you what the story is about, the first pharagraph gives you a synopsys of the whole story hidding the details, and then the details are shown.

The Newspaper Metaphor

The name should be simple
but explanatory. The name, by itself, should be
sufficient to tell us whether we are in the
right module or not. The topmost parts of the
source file should provide the high-level concep
and algorithms. Detail should increase as we mo
downward, until at the end
we find the lowest level functions and details in
the source file.

Vertical Openness

Each group of lines represents a complete thought. Tho
thoughts should be
separated from each other with blank lines

https://github.com/prestrepoh/Clean-Code-Course/blob/master/4-formatting/class-with-blank-lines

vs

https://github.com/prestrepoh/Clean-Code-Course/blob/master/4-formatting/class-with-no-blank-lines.java

Vertical Distance

Have you ever chased your tail through a class, hopping from one function to the next, scrolling up and down the source file, trying to divine how the functions relate and operate, only to get lost in a rat’s nest of confusion?

Have you ever hunted up the chain of inheritance for the definition of a variable or function?

This is frustrating because you are trying to understand what the system does, but you are spending your time and mental energy on trying to locate and remember where pieces are.

Vertical Distance

Concepts that are closely related should be kep
vertically close to each other [G10].

Closely
related concepts should not be separated into
different files unless you have a very good
reason.

For those concepts that are so closely related th
they belong in the same source file,
their vertical separation should be a measure of
how important each is to the understandability
of the other.

Vertical Distance

Variable Declarations: Variables should be declared as close to their usage as possible.

Because our functions are very short, local variables should appear a the top of each function.

Control variables for loops should usually be declared within the loop statement.

Vertical Distance

Instance variables:

Should be declared at the top of the class. This
should not increase the vertical distance of these variables, because in a well-designed class, they are used by many, if not all, of the methods of the class

Vertical Distance

Dependent Functions:

If one function calls another, they should be vertically close,
and the caller should be above the callee, if at all possible

Vertical Distance

Consceptual Affinity:

Certain bits of code want
to be near other bits. They have a certain
conceptual affinity. The stronger that
affinity, the
less vertical distance there should be
between
them.

Affinity
might be caused because a group of functions
perform
a similar operation

Vertical Distance

Consceptual Affinity:

assertTrue() and assertFalse() don't call eachothe, but they have a string affinity, because they share a common naming scheme and perform variations of the same basic task

Vertical Ordering

In general we want function call dependencies to point in the downward direction. That is,
a function that is called should be below a function that does the calling.

This creates a
nice flow down the source code module from high level to low level.

As in a newspaper, we expect low-level details to come last.

Horizontal Formatting

We should strive to keep our lines short.

The 80 lines limit is arbitrary, but we should try to keep our lines short.

Horizontal Formatting

Horizontal Openness and Density:
surround the assignment operators with white space to accentuate them.

Don't put spaces between the function names and the opening
parenthesis.

Use whitespace to accentuate the precedence of operators.

Horizontal Formatting

Horizontal Alignment:

Don't Align code like this:

```
private Socket socket;
private InputStream input;
private OutputStream output;
private Request request;
```

or

```
this.context = context;
socket = s;
input = s.getInputStream();
output = s.getOutputStream();
```

Indentation

```
public CommentWidget(ParentWidget parent, String text) {
    super(parent, text);
}
```

Is better than

```
public CommentWidget(ParentWidget parent, String text)
{super(parent, text);}
```

Team Rules

A team of developers should agree upon a single formatting style, and then every member of that team should use that style.

We want the software to have a consistent style. We don’t want it to appear to have been written by a bunch of disagreeing individuals.