

### Hints that the code you're reading is a mess

#### Rigidity

No change is trivial, every change in the code add more twists and tangles.

#### Complexity

As above, no change is trivial and requires a lot of research.

#### Fragility

Changes breaking other parts of the code.

#### Immobility

You cannot reuse part of the existing code

### General Rules

Follow the **Boy Scout Rule** : Leave the code cleaner than when you found it

Follow the **Principle of Least Surprise**

Follow **Standard Conventions**, both language related and team related

Keep it **simple** stupid

**Don't repeat** yourself

Be **consistent**

Do not override safeties

### Design Rules

Functions should descend **only one level of abstraction**, and statements in a function should be at **the same level of abstraction**

Use **dependency injection**

Keep your **boundaries clean**

**Encapsulate conditionals**, try to avoid negative conditionals

Make logical dependencies physical

Use **polymorphism** instead of if / else or switch / case

Avoid hidden temporal couplings

### Design Rules (cont)

Keep configurable data (ie: constants) at high levels, they should be **easy to change**

Use Enums over constants

### Source Code Structure

Use **vertical formatting** to separate your code and different concepts, you should read your code **from top to bottom** without "jumping" over functions

Variables should be **declared as close to their usage** as possible

**Instance variables** should be declared at the **top of the class**

Put static methods on top of the package

**Similar and dependent** functions should be **close vertically**

Balance between **vertical openness** and **vertical density**. Same rules apply for horizontal density

Do not align your code horizontally

Use **consistent indentation**

### Naming Rules

Use **descriptive** and **intention-revealing** variable names

Make **meaningful distinctions**

Use pronounceable and **searchable names**

Avoid disinformation and encoded names

Avoid member prefixes or types information (Hungarian Notation)

Avoid mental mapping

Replace Magic Numbers with **Constants**

### Functions

**Functions should do one thing and they should do it well**

Functions should be relatively **small**

Functions should have **descriptives names**

Functions should have as **few arguments** as possible (no more than 3 if possible)

Functions should have **no side effects**

Use **explanatory variables** to explain your intent / algorithm

Don't use flag arguments

Avoid output arguments, they're misleading

### Objects VS Data Structures

Data structures **exposes data and have no behavior**.

So, procedural code makes it **easy to add new function** without changing the existing data structures.

Objects **expose behavior and hide data**.

Object Oriented code makes it **easy to add new classes** without changing existing functions

Avoid hybrids (half object and half data structure)

**The Law of Demeter** : A class should not know about the innards of the objects it manipulates. Objects should not expose their internals.

Same as functions : they should **do one thing** and they should be **small**

Avoid and split Train Wrecks : `object A.g etB ().g et C().ge tD();`

Keep the number of instance variables low, if your class have too many instance variable, **then it is probably doing more than one thing**



By CosteMaxime

Published 13th February, 2019.

Last updated 14th February, 2019.

Page 1 of 2.

Sponsored by **CrosswordCheats.com**

Learn to solve cryptic crosswords!

<http://crosswordcheats.com>

### Error handling

Error handling is one thing, **don't mix error handling and code**

Use **Exceptions** instead of returning error codes

Write the try-catch-finally statement first, it will help you structure your code

Don't return null, don't pass null either

Throw exceptions **with context**

### Tests

**F.I.R.S.T** : Fast, Independent, Repeatable, Self-Validating, Timely

**One assert** per test

Keep your tests **as clean as your production code**, they should be easily readable

Use a **coverage tool**

Tests should be **easy to run**

### TDD

**3 Laws of Test Driven Development**, this should ensure that you write your tests and your code **simultaneously**

You may not write production code until you have written a failing unit test

You may not write more of a unit test than is sufficient to fail, and not compiling count as failing

You may not write production code that is sufficient to pass the currently failing test

### Comments

When to write a comment ?

**Explain yourself in code, not in comment.** If it's not possible, take your time to write a GOOD comment.

What makes up a Good comment ?

Use comments to **inform, explain, clarify**, or **warn** the reader

### Comments (cont)

Comment-out code ?

**DELETE IT**

Avoid using more than one language in a single source file (Html comments, Javadoc for nonpublic code)

Avoid inappropriate Informations (change history, license, ...)

Avoid misleading or noise comments

Don't be redundant (`i++; // increment i`)

Closing brace comments (`} // end of function`)

### Credits

From "Clean Code" by Robert C. Martin

Inspired by [this](#) summary

By Coste Maxime



By CosteMaxime

[cheatography.com/costemaxime/](http://cheatography.com/costemaxime/)

Published 13th February, 2019.  
Last updated 14th February, 2019.  
Page 2 of 2.

Sponsored by [CrosswordCheats.com](http://CrosswordCheats.com)  
Learn to solve cryptic crosswords!  
<http://crosswordcheats.com>