CAPHYON

Open4Tech
2020

Code 💎, not 🗑

Or how to write **high quality**, **clean** and **easy to maintain code**.

Writing performant and bug free code is the holy grail of every software developer.
But does the **code quality** matter?

This session is meant to walk you through becoming a better developer,
by simply using some basic principles meant to make your
code shine like gold.

Andrei Zuica
Full Stack Developer

https://cutt.ly/codegold

Source: youtube.com/watch?v=V7PLxL8jll8

Table of contents

2. Coding Standards

A set of rules about the programming style of a specific language

- Naming conventions
- Indentation
- Declarations
- Statements
- Organization
- Programming principles

Importance:

- Easier to integrate new teammates
- Easy to maintain
- Readable
- Time effective

*For **private** projects, there is nothing wrong in creating a different coding convention.*
*For **public** projects, always use the standards.*

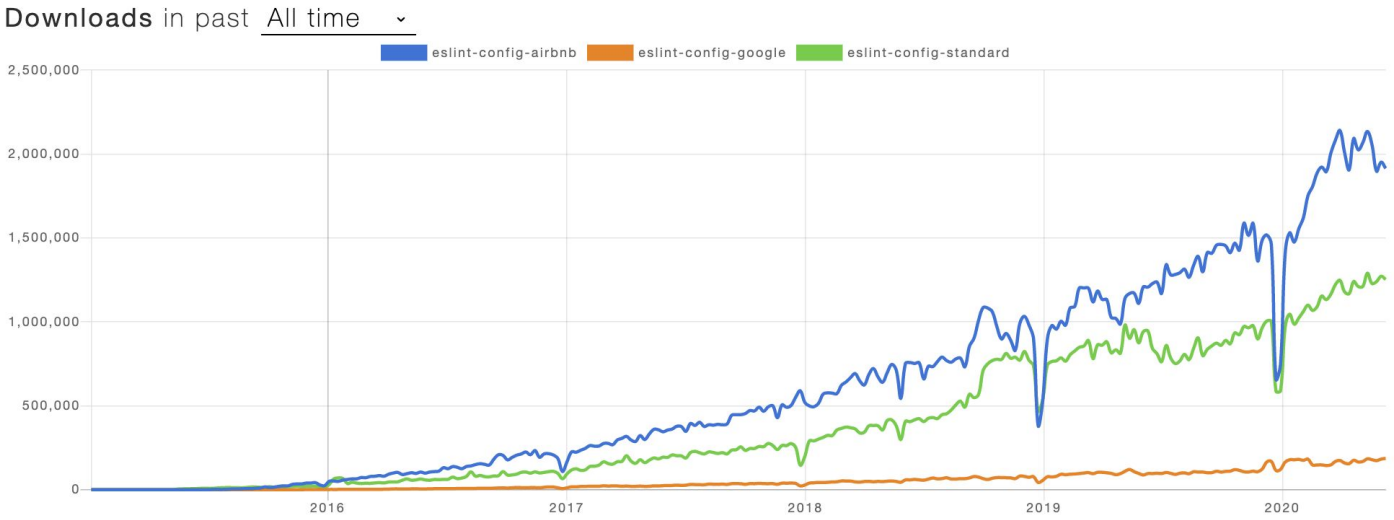
2. Coding Standards - Java example

👎 Bad

```
if (count == 1)
{
//      statements;
} else
if (count == 2) {
        //      statements;
} else if (count > 3 && count <= 100) {
//      statements;
} else
{
//      statements;
}
```

👍 Good

```
if (count == 1) {
//      statements;
} else if (count == 2) {
//      statements;
} else if (count > 3 && count <= 100) {
//      statements;
} else {
//      statements;
}
```



Stats

	stars	forks	issues	updated	created	size
eslint-config-airbnb	97,145	18,896	116	Jun 11, 2020	Nov 2, 2012	minzipped size 301 B
eslint-config-google	1,305	128	9	Jan 13, 2020	Nov 2, 2015	minzipped size 1009 B
eslint-config-standard	1,900	479	17	May 25, 2020	May 13, 2015	minzipped size 2.2 KB

Source: npmtrends.com

2. Coding Standards - Javascript example

2. Coding Standards - Javascript example

AirBnB

```
const params = {
 action,
 url,
 device,
};

let hasParams = true;
Object.values(params).forEach(() => {
 if (hasParams) {
   hasParams = true;
 }
});
```

Standard

```
const params = {
 action
 url
 device
}

let hasParams = true
Object.values(params).forEach(() => {
 if (hasParams) {
   hasParams = true
 }
});
```

3. Code design

```
public List<Integer> getDocSectionsChanged(CustomerVersionTag versionTag) {
        Set<Integer> sections = new HashSet<>();
        for (Map.Entry<String, List<String>> entry : getVersionChanges().get(versionTag).entrySet()) {
                for (F.Tuple<CustomerVersioningDocSection, Map<String, List<String>>> tuple : getDocSectionToSdSection()) {
                        for (Map.Entry<String, List<String>> entry2 : tuple._2.entrySet()) {
                                if (entry.getKey().startsWith(entry2.getKey())) {
                                        for (String change : entry.getValue()) {
                                                for (String lookFor : entry2.getValue()) {
                                                        if (change.startsWith(lookFor)) {
                                                                sections.add(getDocSectionNumber(tuple._1));
                                                        }
                                                }
                                        }
                                }
                        }
                }
        }
        return sections.stream().sorted(Integer::compareTo).collect(Collectors.toList());
}
```

Source: **thedailywtf.com**

3. Code design

1. Organization
2. Indentation
3. Naming conventions
4. Number of arguments
5. Number of members
6. Class too big
7. Dead code
8. Duplicates
9. Comments

Source: **thedailywtf.com**

4. Linters & Formatters

Linter

A software that analyzes code for potential errors.

All modern linters come with warnings about syntax errors, uses of undeclared variables, calls to deprecated functions, spacing and formatting conventions, but they will never resolve those issues. You need to take care of them yourself.

List of linters:
https://github.com/collections/clean-code-linters

Code Formatter

A software that formats your code based on a coding standard.

Some can be configured to format in realtime, ar you may choose to trigger the formatting process yourself, whenever you want.

List of code formatters:
https://github.com/rishirdua/awesome-code-formatters

5. The SOLID principles

S - Single-responsibility principle

"A class should only have a single responsibility"

O - Open–closed principle

"Classes, modules, functions, etc. should be open for extensions, but closed for modification"

L - Liskov substitution principle

"Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program."

I - Interface segregation principle

"Split your interfaces into smaller ones, rather than fewer, bigger interfaces."

D - Dependency Inversion Principle

- *"High level modules should not depend upon low level modules. Both should depend upon abstractions."*
- *"Abstractions should not depend upon details. Details should depend upon abstractions."*

6. Design Patterns

A reusable, generally used solution to a frequent problem within a given application context.

Using **Singleton,** you can easily clean and shorten your code**:**

Classic:

```
ApplicationProperties properties = new ApplicationProperties();
properties.Load();
properties.GetProperties("database.user");
```

Singleton:

```
ApplicationPropertiesSingleton.GetInstance().GetIntProperty("database.user");
```

Using **Builder,** you can easily get rid of too many arguments sent to constructors**:**

Constructor:

```
new Vehicle("Tesla", "CyberTruck", 2020, "Electric", 1400);
```

Builder:

```
new Vehicle.Builder("Tesla", "CyberTruck")
    .withFabricationYear(2020)
    .withEngineType("Electric")
    .withEngineCapacity(1400);
```