Hello!

I am felipe jhordan

Jr. Developer | Catalog

Software Design Principles: DRY, YAGNI, KISS

Software Design Principles

Software design principles are a set guidelines that helps developers to make a good system design.

The list of most importants principles are:

- SOLID principles
- YAGNI
- DRY
- KISS
- Encapsulation

Some questions about a Good/Bad design

Characteristics	Good Design	Bad Design
Change	Change is one part of the system does not always require a change in another part of the system.	One conceptual change requires changes to many parts of the system.
Logic	Every piece of logic has one and one home.	Logic has to be duplicated.
Nature	Simple	Complex
Cost	Small	Very high
Link	The logic link can easily be found.	The logic link cannot be remembered.
Extension	System can be extended with changes in only one place.	System cannot be extended so easily.

1

YAGNI

You ain´t gonna need it - você não vai precisar disso

YAGNI

Practice in software developer which states that features should only be added when required.

YAGNI trims away excess and inefficiency artefacts (code) in development to facilitate the desired increased frequency of releases.

Simplifying...

YAGNI helps avoid spending time on features that may not be used, the main features of a program are better developed and less total time is spent on each release.

YAGNI - wrong example

```
const yagniWrongExample = {
    validate: () => {

        const validateEmail = ({ email }: User) => {
            const AT_FIRST_OCCURRENCE = 2
            const hasAtInEmail = email.indexOf('email', AT_FIRST_OCCURRENCE) == -1

            return hasAtInEmail
        }

        const validateName = ({ name }: User) => {
            const BEST_NAME = 'Jhordan'
            const upperBestName = BEST_NAME.toUpperCase()
            const isBestName = upperBestName === name.toUpperCase()

            return isBestName
        }

        const validateAge = ({ age }: User) => {
            const OF_AGE = 18
            const isOfAge = age >= OF_AGE

            return isOfAge
        }

        const validationsInUse: Function[] = [validateEmail, validateAge]

        const user: User = {
            age: 2,
            email: "mockemail@uol.com",
            name: "Jhordan"
        }


        return validationsInUse.reduce((previousValue, validationFn) => {
            return validationFn(user)
        }, true);

    }
}
```

YAGNI - ok example

```
const yagniBestExample = {
    validate: () => {

        const validateEmail = ({ email }: User) => {
            const AT_FIRST_OCCURRENCE = 2
            const hasAtInEmail = email.indexOf('email', AT_FIRST_OCCURRENCE) == -1

            return hasAtInEmail
        }

        const validateAge = ({ age }: User) => {
            const OF_AGE = 18
            const isOfAge = age >= OF_AGE

            return isOfAge
        }

        const validationsInUse: Function[] = [validateEmail, validateAge]

        const user: User = {
            age: 2,
            email: "mockemail@uol.com",
            name: "Jhordan"
        }


        return validationsInUse.reduce((previousValue, validationFn) => {
            return validationFn(user)
        }, true);


    }
}
```

YAGNI - both examples

```typescript
const yagniBestExample = {
    validate: () => {

        const validateEmail = ({ email }: User) => {
            const AT_FIRST_OCCURRENCE = 2
            const hasAtInEmail = email.indexOf('email', AT_FIRST_OCCURRENCE) == -1

            return hasAtInEmail
        }

        const validateAge = ({ age }: User) => {
            const OF_AGE = 18
            const isOfAge = age >= OF_AGE

            return isOfAge
        }

        const validationsInUse: Function[] = [validateEmail, validateAge]

        const user: User = {
            age: 2,
            email: "mockemail@uol.com",
            name: "Jhordan"
        }


        return validationsInUse.reduce((previousValue, validationFn) => {
            return validationFn(user)
        }, true);


    }
}
```

```typescript
const yagniWrongExample = {
    validate: () => {

        const validateEmail = ({ email }: User) => {
            const AT_FIRST_OCCURRENCE = 2
            const hasAtInEmail = email.indexOf('email', AT_FIRST_OCCURRENCE) == -1

            return hasAtInEmail
        }

        const validateName = ({ name }: User) => {
            const BEST_NAME = 'Jhordan'
            const upperBestName = BEST_NAME.toUpperCase()
            const isBestName = upperBestName === name.toUpperCase()

            return isBestName
        }

        const validateAge = ({ age }: User) => {
            const OF_AGE = 18
            const isOfAge = age >= OF_AGE

            return isOfAge
        }

        const validationsInUse: Function[] = [validateEmail, validateAge]

        const user: User = {
            age: 2,
            email: "mockemail@uol.com",
            name: "Jhordan"
        }


        return validationsInUse.reduce((previousValue, validationFn) => {
            return validationFn(user)
        }, true);

    }
}
```

2

Dry

Don't repeat yourself

Duplication Is Evil

DRY

Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.

Resuming... Divide your code and logic into smaller reusable units and use that code by calling it where you want.

DRY

Instead YAGNI principle, you use the code created, but you remove duplicate sources that can placed in single location.

This shortens the code, more care, if done the wrong way can violate another principle, and guess what? KISS

DRY - wrong example

```
const dryWrongExamples = {
    countAInText: () => {
        const LETTER_TARGET = 'A'
        const lowerLetterTarget = LETTER_TARGET.toLowerCase()

        const lowerText = text.toLowerCase()

        let result = 0
        for (const letter of lowerText) {
            if (letter === lowerLetterTarget) {
                result++
            }
        }

        return result
    },

    countBInText: () => {
        const LETTER_TARGET = 'B'
        const lowerLetterTarget = LETTER_TARGET.toLowerCase()

        const lowerText = text.toLowerCase()

        let result = 0
        for (const letter of lowerText) {
            if (letter === lowerLetterTarget) {
                result++
            }
        }

        return result
    }
}
```

DRY - ok example

```
const dryBetterExamples = {
    countLetterInText: (letterTarget: string) => {
        const lowerLetterTarget = letterTarget.toLowerCase()

        const lowerText = text.toLowerCase()

        let result = 0
        for (const letter of lowerText) {
            if (letter === lowerLetterTarget) {
                result++
            }
        }

        return result
    },


}
```

DRY - both examples

```
const dryWrongExamples = {
    countAInText: () => {
        const LETTER_TARGET = 'A'
        const lowerLetterTarget = LETTER_TARGET.toLowerCase()

        const lowerText = text.toLowerCase()

        let result = 0
        for (const letter of lowerText) {
            if (letter === lowerLetterTarget) {
                result++
            }
        }

        return result
    },

    countBInText: () => {
        const LETTER_TARGET = 'B'
        const lowerLetterTarget = LETTER_TARGET.toLowerCase()

        const lowerText = text.toLowerCase()

        let result = 0
        for (const letter of lowerText) {
            if (letter === lowerLetterTarget) {
                result++
            }
        }

        return result
    }
}
```

```
const dryBetterExamples = {
    countLetterInText: (letterTarget: string) => {
        const lowerLetterTarget = letterTarget.toLowerCase()

        const lowerText = text.toLowerCase()

        let result = 0
        for (const letter of lowerText) {
            if (letter === lowerLetterTarget) {
                result++
            }
        }

        return result
    },


}
```

3

KISS

Keep it simple, stupid!

Simplicity is the ultimate sophistication

Leonardo Da Vinci´s

KISS

The KISS principle is descriptive to keep the code simple and clear, making it easy to understand.

After all, programming languages are for humans to understand — computers can only understand 0 and 1 (I.A can understand both...)

KISS

Simplify the code, but in an organized way.

Maybe thinking, is my code good and simple enough for an intern or even a junior with less experience to understand the meaning?

Avoid mental mappings in the code (Clean Code)

KISS - examples

Wrong

```
const faculty = (n: number): number =>
    n <= 1 ? 1 : n * faculty(n - 1)
```

Better

```
const faculty = (number: number): number => {
    const isSmallerThanOne = number <= 1
    if (isSmallerThanOne) {
        return 1
    }

    return number * faculty(number - 1)
}
```

KISS - Examples

What is better, considering

The kiss principle?

```typescript
const operationException = (msg: string) => {
    throw new Error(msg)
}

function weekday1(day: number) {
    switch (day) {
        case 1:
            return "Monday";
        case 2:
            return "Tuesday";
        case 3:
            return "Wednesday";
        case 4:
            return "Thursday";
        case 5:
            return "Friday";
        case 6:
            return "Saturday";
        case 7:
            return "Sunday";
        default:
            operationException("day must be in range 1 to 7");
    }
}
function weekday2(day: number) {
    if ((day < 1) || (day > 7)) operationException("day must be in range 1 to 7");
    const days: string[] = [
        "Monday",
        "Tuesday",
        "Wednesday",
        "Thursday",
        "Friday",
        "Saturday",
        "Sunday"
    ];
    return days[day - 1];
}
```

KISS - Examples

Sometimes the basic and simple can be more performative, or most of the time.

Setup HTML - click to add setup HTML

Setup JS - click to add setup JavaScript

weekday1

previous run

40632707.43 ops/s ± 2.65%

Fastest

```
"use strict";
const operationException = (msg) => {
    throw new Error(msg);
};
function weekday1(day) {
    switch (day) {
        case 1:
            return "Monday";
        case 2:
            return "Tuesday";
        case 3:
```

weekday2

finished

38359125.79 ops/s ± 3.89%

5.6 % slower

```
        operationException("day must be in range 1 to 7");
    const days = [
        "Monday",
        "Tuesday",
        "Wednesday",
        "Thursday",
        "Friday",
        "Saturday",
        "Sunday"
    ];
    return days[day - 1];
```

CREDITS

- https://www.techtarget.com/whatis/definition/You-arent-gonna-need-it
- https://dzone.com/articles/software-design-principles-dry-and-kiss
- https://www.plutora.com/blog/understanding-the-dry-dont-repeat-yourself-principle
- https://www.freecodecamp.org/news/keep-it-simple-stupid-how-to-use-the-kiss-principle-in-design/
- https://blog.matheuscastiglioni.com.br/yagni/
- https://www.dotnettricks.com/learn/designpatterns/different-types-of-software-design-principles
- https://tateeda.com/blog/fundamental-principles-of-good-software-design
- https://www.dotnettricks.com/learn/designpatterns/different-types-of-software-design-principles

Thanks!

ANY QUESTIONS?

you know where to find me