Clean Code principles

Effectiveness, Efficiency and Simplicity

- **Effectiveness**

First, our code should be effective, meaning it should solve the problem it's supposed to solve. Of course this is the most basic expectation we could have for our code, but if our implementation doesn't actually work, it's worthless to think about any other thing.

- **Efficiency**

Second, once we know our code solves the problem, we should check if it does so efficiently. Does the program run using a reasonable amount of resources in terms of time and space? Can it run faster and with less space?

- **Simplicity**

And last comes simplicity. It avoids unnecessary complexity, intricate logic, or clever tricks that can make code difficult to understand. Strive for code that is straightforward, easy to read, and conveys its purpose without ambiguity.

This is the toughest one to evaluate because its subjective, it depends on the person who reads the code.

Format and Syntax

Using consistent formatting and syntax throughout a codebase is an important aspect of writing clean code. This is because consistent formatting and syntax make the code more readable and easier to understand.

When code is consistent, developers can easily identify patterns and understand how the code works, which makes it easier to debug, maintain, and update the codebase over time. Consistency also helps to reduce errors, as it ensures that all developers are following the same standards and conventions.

Some of the things we should think about regarding format and syntax are:

- **Indentation and spacing**

```
// bad indentation and spacing
const myFunc=(number1,number2)=>{
const result=number1+number2;
return result;
}

// good indentation and spacing
const myFunc = (number1, number2) => {
    const result = number1 + number2
    return result
}
```

- **Consistent syntax**

```
// Arrow function, no colons, no return
const multiplyByTwo = number => number * 2

// Function, colons, return
function multiplyByThree(number) {
    return number * 3;
}
```

Here we have very similar functions implemented with different syntax. The first one is an arrow function, with no colons and no return, while the other is a common function that uses colons and a return.
Both work and are just fine, but we should aim to always use the same syntax for similar operations, as it becomes more even and readable along the codebase.

- **Consistent case conventions**

```
// camelCase
const myName = 'John'
// PascalCase
const MyName = 'John'
```

```
// snake_case
const my_name = 'John'
```

Same goes for the case convention we choose to follow. All of these work, but we should aim to consistently use the same one all through our project.

Naming

Naming variables and functions clearly and descriptively is an important aspect of writing clean code. It helps to improve the readability and maintainability of the codebase. When names are well-chosen, other developers can quickly understand what the variable or function is doing, and how it is related to the rest of the code.

Conciseness vs Clarity

When it comes to writing clean code, it's important to strike a balance between conciseness and clarity. While it's important to keep code concise to improve its readability and maintainability, it's equally important to ensure that the code is clear and easy to understand. Writing overly concise code can lead to confusion and errors, and can make the code difficult to work with for other developers.

Reusability

- Code reusability is a fundamental concept in software engineering that refers to the ability of code to be used multiple times without modification.
- The importance of code reusability lies in the fact that it can greatly improve the efficiency and productivity of software development by reducing the amount of code that needs to be written and tested.
- By reusing existing code, developers can save time and effort, improve code quality and consistency, and minimize the risk of introducing bugs and errors. Reusable code also allows for more modular and scalable software architectures, making it easier to maintain and update codebases over time.

Clear Flow of Execution

- Having a clear flow of execution is essential for writing clean code because it makes the code easier to read, understand, and maintain. Code that follows a clear and logical structure is less prone to errors, easier to modify and extend, and more efficient in terms of time and resources.
- On the other hand, spaghetti code is a term used to describe code that is convoluted and difficult to follow, often characterized by long, tangled, and unorganized code blocks. Spaghetti code can be a result of poor design decisions, excessive coupling, or lack of proper documentation and commenting.

Single Responsibility Principle

The Single Responsibility Principle advises that a class, function, or module should have only one reason to change. , or in other words, each entity in our codebase should have a single responsibility. By separating different responsibilities, clean code becomes more modular, focused, and easier to understand, test, and maintain.

By applying SRP, we can create code that is easier to test, reuse, and refactor, since each module only handles a single responsibility. This makes it less likely to have side effects or dependencies that can make the code harder to work with.

Having a "Single Source of Truth"

Having a "single source of truth" means that there is only one place where a particular piece of data or configuration is stored in the codebase, and any other references to it in the code refer back to that one source. This is important because it ensures that the data is consistent and avoids duplication and inconsistency.

Only Expose and Consume Data You Need

- One important principle of writing clean code is to only expose and consume the information that is necessary for a particular task. This helps to reduce complexity, increase efficiency, and avoid errors that can arise from using unnecessary data.
- When data that is not needed is exposed or consumed, it can lead to performance issues and make the code more difficult to understand and maintain.
- Suppose you have an object with multiple properties, but you only need to use a few of them. One way to do this would be to reference the object and the specific properties every time you need them. But this can become verbose and error-prone, especially if the object is deeply nested. A cleaner and more efficient solution would be to use object destructuring to only expose and consume the information you need.

Modularization

Modularization is an essential concept in writing clean code. It refers to the practice of breaking down large, complex code into smaller, more manageable modules or functions. This makes the code easier to understand, test, and maintain.

Using modularization provides several benefits such as:

Re-usability: Modules can be reused in different parts of the application or in other applications, saving time and effort in development.

Encapsulation: Modules allow you to hide the internal details of a function or object, exposing only the essential interface to the outside world. This helps to reduce coupling between different parts of the code and improve overall code quality.

Scalability: By breaking down large code into smaller, modular pieces, you can easily add or remove functionality without affecting the entire codebase.

Folder Structures

- Choosing a good folder structure is an essential part of writing clean code. A well-organized project structure helps developers find and modify code easily, reduces code complexity, and improves project scalability and maintainability.
- On the other hand, a poor folder structure can make it challenging to understand the project's architecture, navigate the codebase, and lead to confusion and errors.

Keep Comments Relevant and Minimal

While comments have their place, clean code should be self-explanatory and readable without relying heavily on comments. Use comments sparingly, focusing on providing additional context or explaining complex decisions that cannot be easily understood from the code itself.

Write Comprehensive and Meaningful Tests

Clean code is accompanied by thorough and meaningful tests. Tests provide confidence in the behavior and correctness of the code. Well-designed tests cover different scenarios, edge cases, and potential failure points, serving as documentation and aiding in future changes.

By following these principles, developers can produce code that is easier to read, understand, and maintain. Clean code reduces the likelihood of bugs, enhances collaboration, and leads to more efficient and high-quality software development.