Samuel James Follow
Software Engineer @ Venture Garden Group, I speak NodeJs, PHP & VueJs (https://samuelabiodun.com)
Dec 1 · 5 min read

Keeping your code clean

Credit : Mosh

I settled down in my sit, cranking the solution with my team members. “We must win this” I said, burying myself deep down to develop a working prototype within the given two days—The competitive nature in every human had just been ignited and guys must simply compete for 1st, 2nd, and 3rd place.

Some minutes later, one of the judges who was a senior engineer walked up to my desk with a look of disapproval on her face and muttered: “Your code is not clean, It’s messy!” and that was the beginning of my journey towards a cleaner code.

credit: emegen.com

Clean Code? Well, it was not strange to me but does it really matter if the code just works? YES, it does matter, a thousand times.

I had been a software engineer for a few years before this event; I had built working applications, yet I had just been told something different than getting my code to work.

My problem was simple: I focused on getting the job done with an intent of writing code that works and in turn incurring a technical debt that would have to be paid off at some point.

How to code your way to cleanliness

It would not happen the moment you finish reading all the chapters in Clean Code by uncle Bob . It takes knowledge and constant practice, you must learn the principles, patterns and practices. It is hard work that takes years but you can begin today.

No matter how clean a coder you are, there is always one or two things you can learn to become cleaner.

One of the best ways to learn is from the experts themselves by reading their books or posts. You should have their tweets flowing through your twitter feed, listen to talks given by them, follow them on GitHub, study and learn how their code is written and structured.

> *Your growth is limited as an engineer if you do not constantly learn from experts in your field.*

Keep your functions small

This is probably one of the 1,337 articles out there emphasizing on keeping methods or functions as short as possible. One could easily get it wrong here.

> *Clean code is not just about writing short methods but more of writing code that clearly communicates intent.*

When a function is too long, it's likely an indication it's doing too much and readers can get lost decoding what it does. A function should do one thing.

```
if($order->contains($status){
   //do something with order


}
function contains($status){
    $order_statuses=
['accepted','delivered','rejected','processed'];
      if (in_array($status, $order_statuses)) {
            return true;
         }
        return false;
   }
```

We can make function '*contains*' cleaner by re-writing it as:

```
function contains($status){
  return in_array($status, $this->config->statuses);
}
```

Not only is the function '*contains*' cleaner but also decoupled;

Variables or functions should use names that depict what they do

Deciding names to use for functions can be tedious at times but it's definitely worth your effort. It will save you from writing comments that may not get updated when the code changes.

```
$date =date('Y-m-d'); //Ofcourse, it's a date but too
generic!

$orderCreationDate =date('Y-m-d'); //cleaner code
```

Avoid 'If' and 'switch' statements

Personally, it took me a while to grasp this. "How can you tell me to avoid one of my favorites*(if & else)* ?" The truth is, most of the conditional statements can easily be extracted into separate functions and classes. This is not saying you should never use if and switch statements but can be avoided in some cases.
Here is a good example:

```
class BookOrder
{

    public function process()
    {
        switch ($this->book->type) {
            case 'audiocopy':
                return $this->processAudioBook();
                break;
            case 'downloadablecopy':
                return $this->processDownloadableCopy();
                break;
            case 'paperbookcopy':
                return $this->processPaperBookCopy();
                break;
            default:

        }
    }

}
```

A cleaner and a more maintainable way to write this would be:

```
interface  IBookOrder {

    public function process();
}


class AudioBookOrder implements IBookOrder :void {

    public function process()
    {
        // TODO: Implement process() method.
    }
}


class PaperBookOrder implements IBookOrder: void {

    public function process()
    {
        // TODO: Implement process() method.
    }
}


class DownloadableBookOrder implements IBookOrder: void {

    public function process()
    {
        // TODO: Implement process() method.
    }
}
```

Avoid mental mapping

Clean code should be easy to read, understand and should leave no room for guesswork.

> *It is not the language that makes a program look simple, but the programmer who makes the language appear simple.*
> *Robert C. Martin*

The following code checks if a customer can withdraw a certain amount of money. This works, but it's messy.

```
if($this->user->balance  > $amount && $this->user-
>activeLoan===0){
   $this->user->balance -=$amount; // withdraw amount;
}
```

Let’s try to make it cleaner

```
if(!$this->hasActiveLoan() && $this-
>canWithdrawAmount($amount)){
   $this->withdraw($amount);
}

 public function hasActiveLoan(){
    return $this->user->activeLoan===0;
 }
 public function canWithdrawAmount(float $amount){
     return $this->user->balance > $amount;
 }
 public function withdraw(float $amount){
    $this->user->balance -=$amount;

}
```

This is not only easier to understand but also easier to test.

Understand and apply S.O.L.I.D principles

S.O.L.I.D is an acronym for the first five principles of object-oriented programming, defined by Robert C Martin. These principles when put to use, allows you to write code that is low coupled, cohesive and well encapsulated. The principles are closely related and only its successful application can make you write better code. I will save this for another post but you can read about these principles here.

Don’t be too hard on your yourself

Wondering why this is on the list? It’s easy to get caught up in the world of clean code and want to absorb everything once in a day. The sad news is: it takes time, months, years and dedication. The principles must be learned and practiced but it all started with a decision to make everything cleaner, onwards.

Conclusion

By writing this post, I hope to inspire someone to be a better developer and to start caring (*in case you are not*) about clean code.

Feel free to drop your comments, make any of the examples cleaner, add me on LinkedIn or tell me about a new opportunity.

For further reading

Awesome Curated List of Clean Code Resources
Clean Code: A Handbook of Agile Software Craftsmanship 1st Edition
Clean Code PHP
A month of clean code
Clean code concept adapted for PHP
Clean code JavaScript