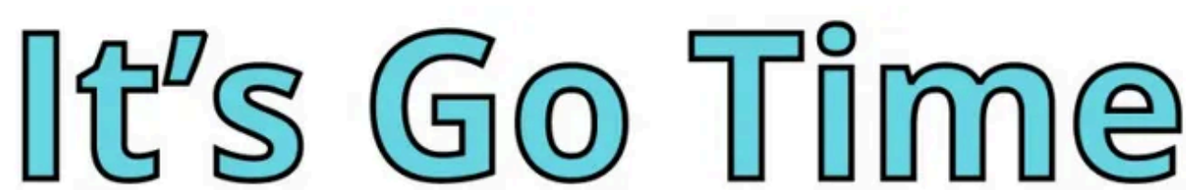
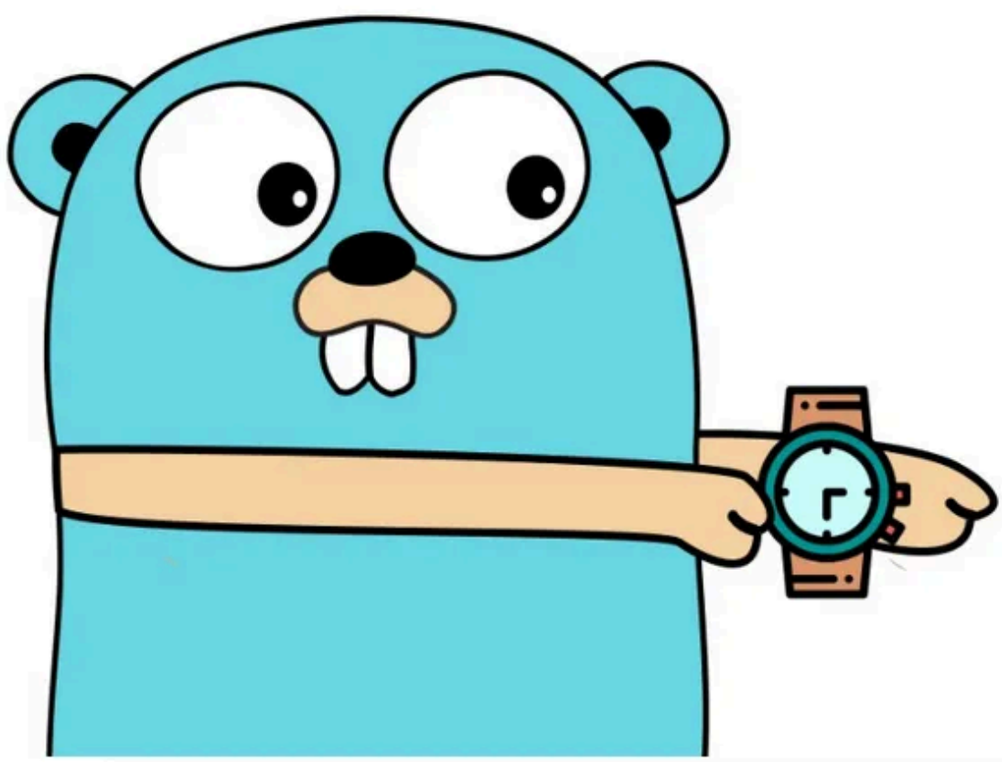
10 Essential Tips for Writing Clean Code in golang

Utkarsh · Follow
Published in Level Up Coding · 6 min read · Mar 10, 2023

Writing clean code is crucial for the success of any software project. Not only does it make code easier to read and understand, but it also makes it more maintainable and reduces the likelihood of bugs. In this article, we will discuss 10 essential tips for writing clean code in Go (golang) that will help you create high-quality, maintainable code.





1. Use clear and concise variable names

Variable names should be clear and concise, indicating what the variable represents. Avoid using single-letter variable names or names that are too generic. A good rule of thumb is to use variable names that are descriptive but not too long. For example:

```
// bad
var x int
```

```
// good
var numEmployees int
```

2. Use meaningful function and method names

Function and method names should be descriptive and accurately describe what the function or method does. Use verbs to describe the action the function or method performs. For example:

```
// bad
func x() {}
```

```
// good
func calculateAverage(nums []float64) float64 {}
```

3. Follow the Single Responsibility Principle

Each function or method should have only one responsibility. If a function or method is doing more than one thing, it becomes harder to read and understand. It also makes it more difficult to test and maintain. For example:

```
// bad
func calculateAverageAndSum(nums []float64) (float64, float64) {}
```

```
// good
func calculateAverage(nums []float64) float64 {}
```

```
func calculateSum(nums []float64) float64 {}
```

> *Like my content? Since, MPP still not supported in India. You can always support by "☕buying me a coffee☕" using the link or qr code https://www.buymeacoffee.com/utkarshjh1*

Encourage me to write more content.

4. Use comments sparingly

Comments should be used to explain why something is done, not how it is done. Code should be self-explanatory, and comments should only be used when necessary. Avoid using comments to explain what the code is doing. For example:

```
// bad
// loop through the array
for i := 0; i < len(arr); i++ {}
```

```
// good
for index := 0; index < len(array); index++ {}
```

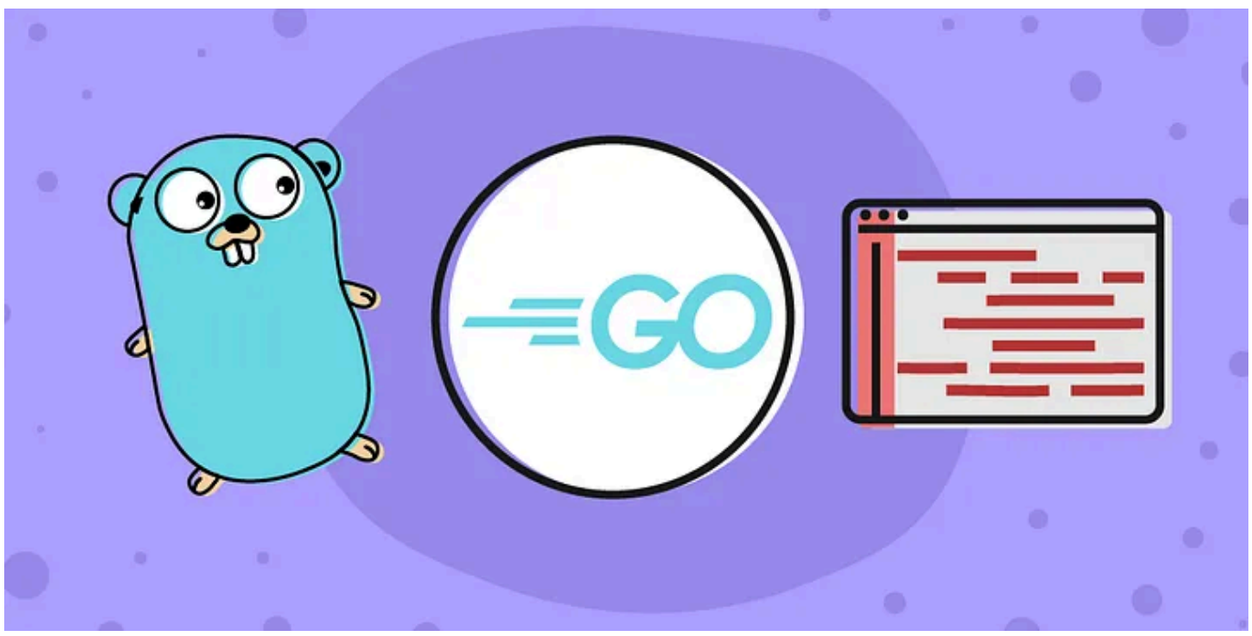
5. Avoid using global variables

Global variables can make code harder to test and maintain, as they can be modified from anywhere in the code. Instead, use local variables or pass variables as parameters to functions or methods. For example:

```
// bad
var count int
```

```
func incrementCount(count int) int {
    return count + 1
}
```

> *Have you already learned a lot? I am curious about the important series of topics we will come across further below. Let's continue our journey further.*



6. Handle errors properly

Always handle errors in your code. Ignoring errors can lead to unexpected behavior and make your code harder to debug. Use Go's built-in error handling mechanism to handle errors in your code. For example:

```go
// bad
f, err := os.Open("filename.txt")
if err == nil {
    // do something with f
}
```

```
// good
f, err := os.Open("filename.txt")
if err != nil {
    log.Fatal(err)
}
defer f.Close()
// do something with f
```

7. Use interfaces to decouple dependencies

Using interfaces can help decouple dependencies in your code. This makes it easier to test and maintain your code. For example:

```go
// bad
func calculateAverage(nums []float64) float64 {
    sum := 0.0
    for _, num := range nums {
        sum += num
    }
    return sum / float64(len(nums))
}
```

```go
// good
type Calculator interface {
    Calculate(nums []float64) float64
}

type AverageCalculator struct {}

func (c *AverageCalculator) Calculate(nums []float64) float64 {
    sum := 0.0
    for _, num := range nums {
      sum += num
    }
   return sum / float64(len(nums))
}
```

8. Use proper formatting

Proper formatting makes your code more readable and easier to understand. Use Go's built-in formatting tools to ensure that your code is properly formatted. For example:

```go
// bad
func calculateAverage(nums []float64) float64 {
sum := 0.0
for _, num := range nums {
sum += num
}
return sum / float64(len(nums))
}
```

```go
// good
func calculateAverage(nums []float64) float64 {
    sum := 0.0
    for _, num := range nums {
        sum += num
```

```
    }
    return sum / float64(len(nums))
}
```

9. Keep functions and methods short

Functions and methods should be short and concise. This makes them easier to read and understand. A good rule of thumb is to keep functions and methods to less than 50 lines of code. If a function or method is longer than that, consider breaking it up into smaller, more manageable functions or methods. For example:

```
// bad
func calculateAverageAndSum(nums []float64) (float64, float64) {
    sum := 0.0
    for _, num := range nums {
        sum += num
    }
    average := sum / float64(len(nums))
    return average, sum
}
```

```
// good
func calculateAverage(nums []float64) float64 {
    sum := 0.0
    for _, num := range nums {
        sum += num
    }
    return sum / float64(len(nums))
}

func calculateSum(nums []float64) float64 {
    sum := 0.0
    for _, num := range nums {
        sum += num
    }
    return sum
}
```

10. Write testable code

Write code that is easy to test. This makes it easier to identify and fix bugs in your code. Use the “testing” package provided by Go to write tests for your code. For example:

```
// bad
func calculateAverage(nums []float64) float64 {
```

```
    sum := 0.0
    for _, num := range nums {
        sum += num
    }
    return sum / float64(len(nums))
}
```

```
// good
func calculateAverage(nums []float64) float64 {
    if len(nums) == 0 {
        return 0.0
    }
    sum := 0.0
    for _, num := range nums {
        sum += num
    }
    return sum / float64(len(nums))
}

func TestCalculateAverage(t *testing.T) {
    nums := []float64{1.0, 2.0, 3.0, 4.0}
    expected := 2.5
    result := calculateAverage(nums)
    if result != expected {
        t.Errorf("Expected %f but got %f", expected, result)
    }
}
```

Conclusion

Writing clean code is essential for creating high-quality, maintainable software. Follow these 10 tips to write clean code in Go that is easy to read, understand, and maintain. By doing so, you will improve the quality of your code, reduce bugs, and make your code more testable.

> *Like my content? Since, MPP still not supported in India. You can always support by "☕**buying me a coffee**☕" using the link or qr code https://www.buymeacoffee.com/utkarshjh1*

Will Encourage me to write more content.

> *I hope you found these **10 tips helpful for writing clean code in Go**. By following these guidelines, you can produce code that is easy to read, maintain, and test, which will ultimately **save you time and effort in the long run**. Remember that writing clean code is an ongoing process, and it requires constant effort and attention. So, keep these tips in mind and strive to make your code as clean and readable as possible.*
>
> *If you enjoyed this article, **please give it a clap and leave a comment with your thoughts or questions**. Your **feedback is always appreciated**, and it helps me create better content in the future. And if you want to see more content like this, be sure to **follow me for future updates**. Thank you for reading, and happy coding!*

Golang | Programming | Technology | Golang Tutorial | New Writers Welcome

Responses (4)

What are your thoughts?

Respond

Eric PASCUAL
almost 2 years ago

I don't see the point with your example related the interfaces. Where is the advertised decoupling in action there compared to the supposedly bad code ?

Maybe you should try something else because as is it doesn't really demonstrate what interfaces are good for and when they should be used.

My $0.02

5 1 reply Reply

Jarrod Roberson
over 1 year ago

Who is paying for all these useless articles that say the same thing that do not say anything that hundreds of other posts have been saying for the last 10 years of Go?

This is the 5th "Clean Code in Go" and all the headings are even the same, the.....

Read More

5 1 reply Reply

Andy Spence
almost 2 years ago

One clarification? The Single Responsibility Principle is widely misconstrued as Do One Thing but that wasn't what Uncle Bob meant. A better quote of his is "Gather together those things that change for the same reason and separate those things that.....

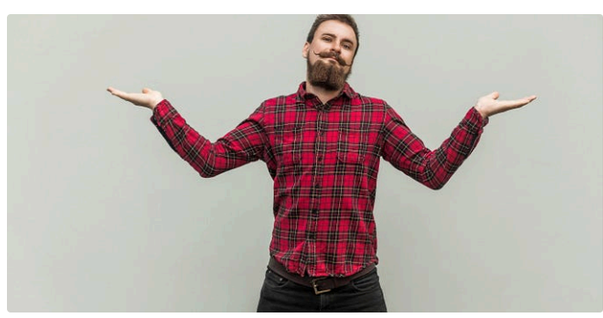
Read More

6 1 reply Reply

See all responses

More from Utkarsh and Level Up Coding

In Level Up Coding by Utkarsh



In Level Up Coding by Marcin Kwiatkowski

Go for DevOps: Automating Infrastructure with Go

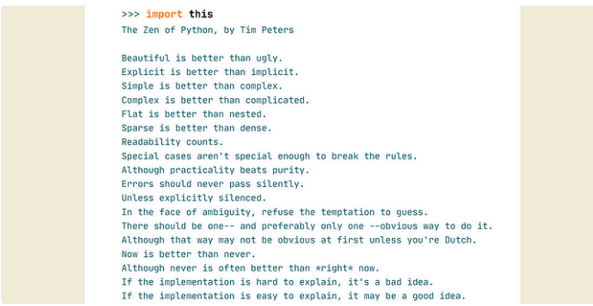
Introduction: DevOps is a set of practices that combines software development and IT...

Mar 31, 2023 437

How Saying ‘I Don’t Know’ Can Make You a Better Software...

For several years, I’ve been working remotely, and one of the most annoying things in my...

1d ago 31 1



In Level Up Coding by Joseph Robinson, Ph.D.

From Messy to Masterpiece: The Art of Pythonic Coding

The One Pep20 Guide You’ll Need + 10 Hacks for Beautiful Code

1d ago 537 6

In Level Up Coding by Utkarsh

Golang vs Python: Battle of the titans

Introduction

Jul 29, 2023 60

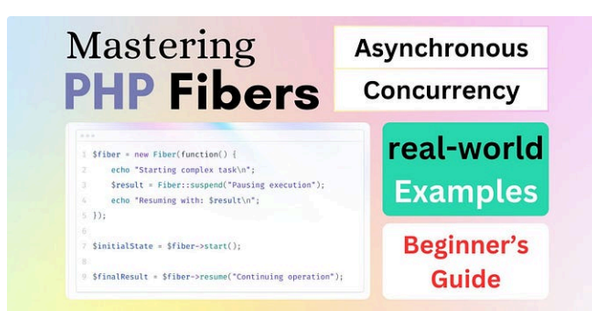
See all from Utkarsh

See all from Level Up Coding

Recommended from Medium

In Level Up Coding by Matt Bentley

My Top 3 Tips for Being a Great Software Architect



Arif Hossen

Mastering PHP Fibers: A Game-Changer in Concurrency...

Revolutionizing Asynchronous Code

My top tips for being a great Software Architect and making the best decisions for...

6d ago 1.2K 16

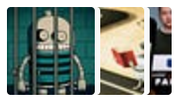
6d ago 20

Lists

General Coding Knowledge
20 stories · 1853 saves

Coding & Development
11 stories · 963 saves

ChatGPT prompts
51 stories · 2442 saves



AI Regulation
6 stories · 669 saves

Harendra

How I Am Using a Lifetime 100% Free Server

Get a server with 24 GB RAM + 4 CPU + 200 GB Storage + Always Free

Oct 26, 2024 8.5K 129

Kenzy Limon

Best Practices when developing Golang Backend Database APIs....

Welcome back to the thrilling finale of our Golang backend development series! 🎉 By...

Jan 2 85 1

Jessica Stillman

Jeff Bezos Says the 1-Hour Rule Makes Him Smarter. New...

Jeff Bezos's morning routine has long included the one-hour rule. New...

Oct 30, 2024 18.7K 475

In Go Golang by Yash

8 Golang Performance Tips I Discovered After Years of Coding

These have saved me a lot of headaches, and I think they'll help you too. Don't forget to...

Oct 17, 2024 816 8

See more recommendations