Is your code rigid and|or fragile?!

Nsovo Manganyi
Problem solver | Senior Software Developer at Yoco | Community Builder
Published May 5, 2017

+ Follow

There is code that is not necessary good, code that just works. As a good developer you want to make sure that your code is readable , understandable and clearly explains the intent of what is being written.

When you modify a function to fix certain bug(s) but you have to change massive amounts of other code and chase the tail through the code it could be an indication that code is bad. Bad dependencies and coupling can make your code rigid.

Single responsibility principle from SOLID principle by Uncle Bob can guide you into writing code that is not rigid or fragile.

Solid principle-: Single responsibility principle

A class should have one reason to change. It is simple theory right? However achieving it can be quite tricky at first. Too many responsibilities on a single thing can cause problems.

When Implementing SRP you are able to reduce coupling, increase cohesion, less fragile code, better testability to mention a few. Having this principle in mind when designing and developing will lead your code to be more readable, simpler and maintainable.

Examples: pseudocode :)

```
public class BankAccount
    {
        public string AccountNumber { get; set; }
        public decimal AccountBalance { get; set; }

        public decimal CalculateInterest()
        {
            // Write logic here
        }
    }
```

Looking at this example it is clear that you have more than one reason of change requests, violates Single Responsibility Principle. For instance you could be asked to implement a

simple change to add *AccountholderName*, and things (interest rates) which is not related to the change request breaks.

The next change request could be add a new rule that has been introduced to calculate interest. In order to solve this problem we could break BankAccount and InterestCalculator into interfaces to be implemented accordingly.

```
public interface IBankAccount
    {
        string AccountNumber { get; set; }
        decimal AccountBalance { get; set; }
    }
 public class BankAccount : IBankAccount
    {
        public string AccountNumber { get; set; }
        public decimal AccountBalance { get; set; }
    }
```

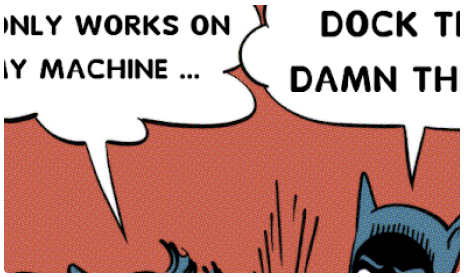
```
public interface IInterstCalculator
    {
        decimal CalculateInterest();
    }

public class InterstCalculator : IInterstCalculator
    {
        public decimal CalculateInterest(IBankAccount account)
        {
            // Write logic here

            return new decimal();
        }
    }
```

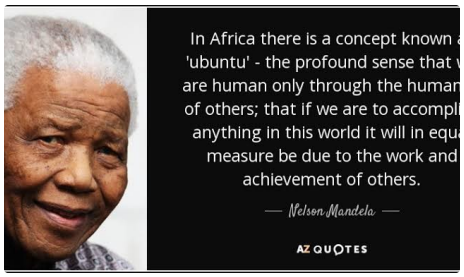
The main aim at the end of the day is to write code that is clean, manageable, reusable and yes easy to read. What is important as a developer is to know that no matter how tight the deadline is you will not make the deadline by making a mess, actual fact is the mess will slow you down instantly and might force you to miss the deadline.

More articles by this author



Create a Docker image for your application

Jul 18, 2021



How can my work be of service ?

Dec 3, 2019

Vertical slicing, My coding journey

Nov 20, 2019

See all →

Insights from the community

Code Review

How do you deal with magic numbers and hard-coded values in your code?

Software Engineering

What are common code smells that indicate poor code quality?

Application Development

How can whitespace and indentation improve code readability?

Algorithms

What are the best ways to document and comment your algorithms for readability and maintainability?