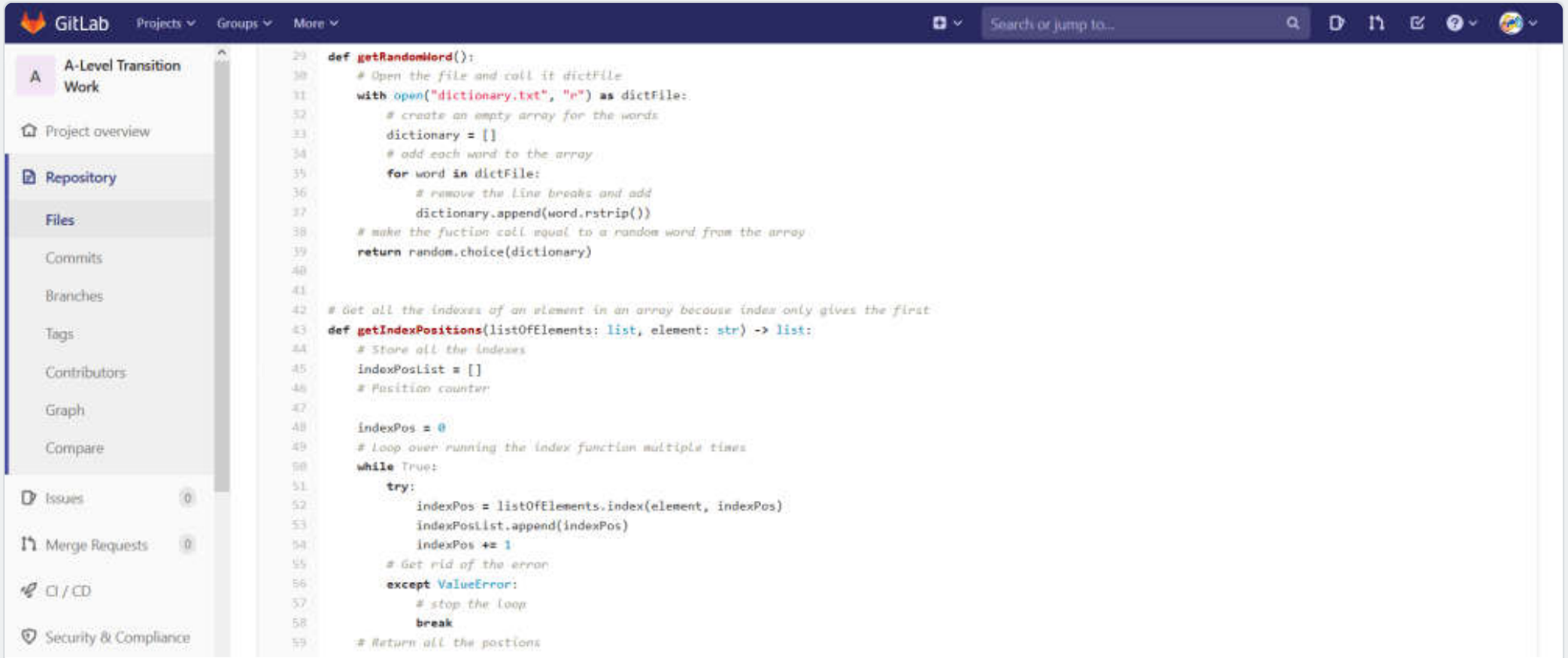

Programming Principles

#computerscience

Richard Jun 21 *Updated on Nov 06, 2020* · 4 min read

I'm not going to hide it when I write any code it tends to get quite messy. By the end, it tends to look pretty bad, so a useful thing to do is refactor code when it's done.

Personally, I like to focus on getting the task done and working efficiently and refactor when it's done. It is crucial to have maintainable and readable code if others will be looking at it. Here are some key principles you should be following:

Keep your code DRY.

Don't repeat yourself. When writing lots of code, you get tired and repeat functionality and styles. Don't. Just add it all into a function or a helper class. Let's say you need to generate HTML elements via Javascript, it is much cleaner to add this to a procedure and call it multiple times; making your code much more readable.

YAGNI

You aren't gonna need it. This principle says you shouldn't be adding functionality until you actually need it. Implementing functionality when you foresee it's usage isn't a good idea because you will end up not needing it and will have unused code.

KISS

Keep it simple stupid. Always make sure you go with the cleanest solution, to make it as readable as possible. One way to increase readability is to use functions in the standard library rather than writing your own lengthy solution. I don't mean download an npm package when you can quickly write something yourself but keep your code sensible. Here is an example in python using a simple sorting algorithm:

```
# Bad

myNumbers = [5,8,1,2,10,21,3]

flag = 0
while not flag:
     if(all(myNumbers[i] <= myNumbers[i + 1] for i in range(len(myNumbers)-1))):
          flag = 1
     for i in range(len(myNumbers)-1):
          _next = i + 1
          if myNumbers[i] > myNumbers[_next]:
               myNumbers[i],myNumbers[_next] = myNumbers[_next],myNumbers[i]
          else:
               continue

print(myNumbers)

# Good

myNumbers = [5,8,1,2,10,21,3]
myNumbers.sort()
print(myNumbers)
```

why make life more complicated than it needs to be?

Writing Code defensively

Defensive programming is not something I tend to do on small projects, but it is concerned with writing code in a way in which it is near enough impossible to break. A programmer would consider *every* single

possible situation and make sure nothing breaks by using something like a try-catch exception (or try-

and-except for the Python fans). Defensive programming involves improving quality and comprehensibility and making software predictable.

"If it ain't broke, don't fix it"

Bert Lance is known for popularising this phrase in 1977. It is just what it says on the tin. Don't try to fix something that isn't broken; you'll probably end up breaking it. Then you'll spend ages debugging (I've been there).

Gall's law

Gall's Law is a rule of thumb for systems design from Gall's book Systemantics: How Systems Really Work and How They Fail. It states:

> "A complex system that works is invariably found to have evolved from a simple system that worked. A complex system designed from scratch never works and cannot be patched up to make it work. You have to start over with a working simple system." [1]

If I were to simplify that for you; it basically says that you should be building on something simple that works to make something big that works, because making something big from scratch will end in mayhem.

Separation of Concerns

The Separation of Concerns (or SoC for short) is a design principle which says you should separate

sections of a computer program so that each is dealing with a separate concern. An example of this is

HTML, CSS and JS you should keep them each to their own file to separate the markup the styles and functionality. This is again useful to improve the comprehensibility of code.

Single-responsibility principle

Similarly, the Single-responsibility Principe states that each module of code should only have responsibility over a single part of the functionality. Let's say you are writing a game. You should separate each part of the game such as movement, speech into separate methods or functions, rather than have one function or method dealing with everything.

Worse is better

This may sound a little strange, but "it is the subjective idea that quality does not necessarily increase with functionality — that there is a point where less functionality ("worse") is a preferable option ("better") in terms of practicality and usability." This principle by Richard P. Gabriel states that if increasing the functionality makes something harder to use then it is better to have less functionality.

Interface segregation principle

The Interface segregation principle (or the ISP) states that "no client should be forced to depend on methods it does not use" [2]. This means that a large interface would be split into smaller section so that the users can find what they need with ease without having to scroll through too many predefined steps (methods).

Ninety-ninety rule

The 90-90 rule states that "The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time." This adds up to 180% of development time alluding to the fact that notoriety of software development projects significantly over-running their schedules.

Hope you learned something new. Feel free to hit the heart button; and follow if you would like to see more of my thoughts.

✨If you would like to follow my day to day development journey be sure to check my Instagram.

Discussion

Subscribe

Add to the discussion

David

Jun 22 •••

This is a lot, Richard -- a bit of an amalgamation of a lot of things. You've summed up a lot of the things that is captured under SOLID by "Uncle Bob" (Robert C. Martin) and the cult of "Clean Code" (maybe not cult).

I'll note that in practice, DRY sometimes does more harm than good -- the abstraction or just some box of functions gets created and re-used everywhere and sometimes used in a "refactor" that modifies existing code. DRY is okay so long as

you're adhering to SOLID principles.

You'll come to learn that some of these are a bit redundant and already captured by SOLID. Separation of concerns already falls under the letter I of SOLID, which stands for interface segregation principle, and to some extent, the letter S, in SOLID for single responsibility principle. There's a great JavaScript repository with examples of what not to do and what to do here: clean-code-javascript

I would probably not state that defensive programming is something we should practice -- most certainly not. I would consider it an anti-pattern and does not lend code to comprehensibility. There's no way writing try/catch everywhere is going to make your code easier to read. Try/catching on a standard error is a red flag saying, "I don't know what behavior my application exhibits, but please don't break!" An experienced developer will learn and understand the difference between exceptions and errors. We should understand the expected behaviors of our abstractions and be explicit about handling them. We should never perform catch-alls...

I practice and review code regularly and every principle from SOLID is exhibited. The open/closed principle is sometimes what is at odds with DRY, because it really suggests that any code that's written and unit-tested is "open for extension" but "closed to modification". This just translates to not modifying/editing/refactoring the code after it is written. Why? Because it literally falls inline with "if it ain't broke, don't fix it." If something is working and you never touch it, it's not going to break. The only time that we should modify any existing file/code is if there is a bug (it's broken). If we need to extend it, we should always create a new version of it and also employ polymorphism and do minimal swaps if we need this new version to replace the old version

5

Reply

Richard

Jun 22

Thankyou for the well explained comment; I will add a little more clarity to some of the points.

2

Reply

David

Jun 22

Keep going, Richard! I love that you're digging into this stuff early on

2

Reply

Lalit Yadav

Jun 25 •••

Nice, here is my favourite one:- "Anything that can go wrong will go wrong" - Murphy's Law

2

Reply

Richard

Jun 25 •••

Loving the pessimism in that quote

2

Reply

Joel Manley

Jun 30 •••

I immediately thought of SASS, which enables this to some degree.

2

Reply

Cole Geerts

Jun 21 •••

I'd agree. I've worked positions where this principle has been taken too far. While an important idea, I can't help but feel it is often miss-applied. Great article otherwise though.

2

Reply

 Richard

Jun 21 •••

Thankyou for the clarification.

1

Reply

Code of Conduct • Report abuse

Richard

On a journey to becoming a Web developer.

Follow

LOCATION
England, UK

JOINED
Oct 5, 2019

Trending on DEV 🔥

 What editor, browser and terminal do you use?

#webdev #watercooler #productivity #beginners

 Why Do Most Programming Languages Count From Zero?

#computerscience #codenewbie #todayilearned #programming

Creating a Coding Interview Study Guide

#javascript #computerscience #beginners #career