Misc

Friday, November 23, 2018

1:16 PM

## Bout Me

L4 GRE Engineer for Target since beginning of June, but I mostly pretend that I'm a Senior Software Engineer and somehow get away with it.  
God-Emperor of ACM UMN

Despite the title I'm actually human, allegedly.

Started coding 17ish years ago for Ultima Online Sphere Server with C#

I still suck at "knowing" languages. I'm more an of algorithms person. I can Google the rest

Started martial 25 years ago

I have the best friends ever  
I'm weird af.

And gay af.

Also completely out of my mind.

Completely unashamed of all of those things.

Yes, I can beat your dad up.

## 

## Modularize That Shit

Automagically makes tools for your tools.

Looks sharp af

Makes testing, and modifying code 724 kajillion times faster and easier

Makes code re-usable for this or other projects (kind of the point, right?)

## Documentation

If your API is callable directly create a --help or --docs parameter for the script.

When your function is called incorrectly it should explain why it was called incorrectly upon failure.

Add a small block comment at the beginning of each file, function, class, etc. explaining what it does and how to utilize it.

Include contact information for users to report bugs.

Variable, class, etc. names should be a type of documentation.

## Use Other Tools and APIs

Linters such as Pylint, Black, Pycharm, etc. will find syntax and style errors as well as other suspect code, as you make those mistakes.

Don't be afraid to use other APIs and tools, however you may occasionally need to keep in mind some restraints

## Be aware of Restraints

Is it okay if your code isn't super fast? Probably, but make sure you are still staying within' a reasonable limit.

Is your software being used on imbedded systems? Then you better not import 800 huge libraries, or require massive amounts of computing power or memory.

Does it have to run on Windows, Mac, and Linux?

Do you really have time to do all of the things I said? Sometimes you have to make sacrifices. By sometimes, I mean always. "With perfect success comes perfect sacrifice." - InspiroBot

## Write Code that Writes Code

It's like writing to a file, but you give it a special file ending.

Yea, it's that simple.

Yea, that shit's as crazy awesome as it sounds.  
Easy for situations when you need many similar functions

Creates a way for less code-savvy folks to write code. E.g.

Navigating Target's mainframe

Recording actions taken on a UI and turning them into backend code

## Style

Listen to your linter

Be consistent. While many languages have standards that will be pointed out by your linter, sometimes you won't have a linter, or the language doesn't have a standard style.

Code reviews

Keep your lines short whenever possible. 80 lines is a good guide, but sometimes things happen.

Keep function lengths short

Default parameters

Avoid Telescoping

## Get Help

Communicate with potential users to get an idea of what tools, functionalities, inputs, outputs, etc., should be included and handled.

Code Reviews help find errors and style issues that might be missed otherwise.

It also helps improve the skill of both the reviewer and reviewee.

Bonus part? It probably doesn't count as cheating in your class.

## Modularize That Shit

Automagically makes tools for your tools.

Looks sharp af

Makes testing, and modifying code 724 kajillion times faster and easier

Makes code re-usable for this or other projects (kind of the point, right?)

Yes, I did need a whole screen just for this.

Q's about Target

Saturday, December 1, 2018

6:44 AM

## What is GRE?

Guest Reliability Engineering is basically Service Reliability Engineering, which is like putting out digital fires, except we can automate the fire extinguisher, or fix the actual problem ourselves, occasionally.

## 

## How can I has job?

Get an internship first if you don't already have experience.

Apply so many times. There's a flood of new engineers.

Have a nice looking resume. Use LaTeX at the very least.

Know the job that you are applying for.

Have outside projects

Get involved with groups like ACM-W, ACM, WIE, SWE, and WIT

Be yourself. If you don't, you will end up with a team you hate.

## Do They Drug Test?

No

## 

## Will You Give Recommendation for me?

I can let someone know that you are particularly interested in a job position. Beyond that, probably not.

Test that shit

Thursday, November 22, 2018

8:22 PM

## Auto-testing

## 

Setuptools looks for any files with \_test or test\_ in them when you run "python3 setup.py test"

Drone.io works much like Setuptools, but it fires from Github instead of locally.

Other automated testing

<https://github.com/atinfo/awesome-test-automation/blob/master/automation-and-testing-as-service.md>

## 

## Mocking

Runs faster.

Avoids Side-effects and rate limits

Typically uses decorators

Only like 13 people actually know how it works.

Generally more useful for classes, as you can modify attributes

## 

## Error Reporting as a Form of Testing

Good error reporting turns users into debuggers. Isn't that nifty?

Add very frequent checks for function success. Too many checks are better than too few.

Keep error text unique and descriptive. Error reporting only helps if you can find the origin of the error

## Log

Use a logger. Just Google logger <language> and you will find one.

Python's logging module is great

Allow the option to log to stdin or file. You can usually do this with parameters or config files

## Minions

Before releasing publicly, get a couple early adopters to help find errors for you.

Be careful

Friday, November 23, 2018

10:41 AM

## Avoid Side-Effects

## 

Run all checks that could indicate program failure as early as possible in the program.  
When side-effects are unavoidable

Push that portion of the code as far as possible to the end

Try to include code that will undo as many side-effects as possible upon failure

Verify that every step has succeeded before proceeding to the next. (Use decorators?)

## Don't Raise Exceptions

API's should rarely(read never) propagate errors to the calling module. Perfect utilities should be written such that every failure is because of a known reason, that reason can be clearly stated in an error, and that error can be recovered from.

## Clearly Indicate Success/Failure

Success/Failure should be obvious in all function returns. Ideally successful runs should not return falsey values (None, Null, False, [], '', "", etc.). If there is a situation in which your function must return a falsey value for a successful run you should return an additional parameter to indicate success/failure. If you are unable to return an additional parameter be certain that the falsey value returned in the failure case is always different than any falsey value returned in the successful case.