Assignment in Static Test Techniques

Meta Info

Find the code on GitHub at testing-course/week-1 at master · edipetres/testing-course · GitHub

2. Static Code Analysis of the Triangles program

a) Install Metrics software in your IDE
c) Calculate central metrics in your Triangle program

I rewrote my triangles program in Java, but then I could not get the metrics plugin for Netbeans working. Could be a version mismatch with my IDE. Anyway, I found a nice tool for python called `Radon`. It produced the following results:

```
$ radon cc triangles.py -a -nc

triangles.py
    F 3:0 specify_triangle - C (12)

1 blocks (classes, functions, methods) analyzed.
Average complexity: C (12.0)
```

e) Possibly refactor your code based on static testing results. You might also want somebody else to review your code. Write down what changes you make base on static techniques used.
The analysis of my code shows that the cyclomatic complexity of my program is quite high at 12.0 points. Having a CC over 10 means we should refactor our code. To do this we need to reduce the number of if statements in our code.

I replaced an `if` statement checking that the values are positive with an assert function. This should have reduced the CC to 9, but my metrics tool is was acting weird and still show a CC of 12 or even show nothing. I presumed there is a but and I believe this way the CC actually will drop below 10.

b) Check coding standards in your Triangle program

See the outlined coding standards that I checked with my program below in exercise 5. For the exception of the indentation size which I prefer using 2 spaces, the coding practices were mostly right in this small code.

d) Find out what CC variation that your metrics tool uses

The Radon program is using *McCabe's complexity* for CC metrics.

f) Write test cases in xUnit tool

Since my code is not written in one of the languages supported by xUnit I wrote unit tests in Python, with the standard `unittest` package. Find the test cases in the code on Github.

3. Peer Review Checklist

Even the most professional people will miss some important steps when doing task repeatedly.
Checklists are useful for ensuring that all important aspects are considered and not forgotten about. It can also be an ease of mind. Instead of having to meticulously remember every point, we rely on the checklist to ensure everything is done.

One of the most important things is the time aspect of doing reviews. We have to recognize that we are humans and a lot of decisions are made in the brain during a review process. Therefore it is key to keep a fresh mind to be our best when doing the code reviews.

5. List the coding standards - best practices and code conventions - that you find most important for a team to follow

The coding standards recommended for writing Python code are set in stone in the PEP8 document: PEP 8 – Style Guide for Python Code | Python.org.

- **Indent using spaces**
- indent with 4 spaces (or 2) - but stick with one!
- if indented with tab, do not mix space and tab indentations - python3 will disallow this

- **maximum line length** of 79 characters
- for string concatenation stick to 72 chars
- this allows for two file editors be open next to each other without the text overflowing

- **blank lines**
- surround top level functions and class definitions with two blank lines
- method definitions inside class are surrounded by a single blank line
- inside functions use blank lines sparingly to show a logical separation

- **imports**
- imports should be individual in separate lines
- imports are at the top of the file
- imports should be grouped in order:
  - standard library imports
  - related third party libraries
  - local app/lib specific imports

- **module level dunders** e.g. `__version__, __author__` must have the following order
- first string docs
- then **future** imports
- dunder definitions: `__version__ = '0.1'`
- before normal imports

- **String quites**
- string quotes are not enforced, use one and stick with it

6. Highlights from lecture by Gitte Ottosen

I really enjoyed the personal story on testing a critical system as one of her job with an early production release. This story highlighted how testing is used as an indicator for the code quality in companies.

I also found important the notion that developers are responsible for the quality of their own code. Therefore, everyone should be writing tests to ensure their code is of high quality.

A nice recap on Agile Methodologies, emphasizing the value priorities with the keyword **over**.