Software Metrics: Weeks 7&8

Due Sunday 12/01/2019 at 11:59 pm.

Overview:

Complexity analysis of some tokenizer code.

In Blackboard:

1. Part I: Complete the **presentation questionnaires for the presentations**.
2. Part II: Choose one of the complexity measures covered in class (Big O, Halstead, or McCabe). Use this measure to **determine the complexity of tokenizer.py**.

**Part I:** Answer the presentation questions for the guest speakers’ presentations. The presentation questions are in a file on GitHub under [cityuseattle](https://github.com/cityuseattle)/[CS\_504\_FALL\_2019\_ONSITE](https://github.com/cityuseattle/CS_504_FALL_2019_ONSITE). The filename is Presenter questions.docx.

Here are the questions you should answer for each presenter:

* How do you determine requirements?
* How do you communicate within and across groups?
* How do you move from stage to stage in a project? (What stages do you have?)
* How do you ensure quality as you are developing a project?
* How do you test your projects? How do you know it is ready for release?
* How is a product maintained after release?
* What management structure do you have?
* How do you improve your development process?

**Part II:** Complexity: Determine the complexity of tokenizer.py. (Below or on GitHub -- be sure to use the tokenizer.py version on GitHub NOT the tokenizer.py.txt version.)

Be sure to answer the following questions:

1. Which complexity measure did you choose?
2. According to this complexity measure, how complex is the code (YES including the “for token in tokenize” statement that runs the code)?
3. Do you agree with this complexity measure and the complexity of the code? Why or why not?
4. Name at least 3 things you learned in completing this complexity analysis.

tokenizer.py:

import collections

import re

Token = collections.namedtuple('Token', ['type', 'value', 'line', 'column'])

def tokenize(code):

keywords = {'IF', 'THEN', 'ENDIF', 'FOR', 'NEXT', 'GOSUB', 'RETURN'}

token\_specification = [

('NUMBER', r'\d+(\.\d\*)?'), # Integer or decimal number

('ASSIGN', r':='), # Assignment operator

('END', r';'), # Statement terminator

('ID', r'[A-Za-z]+'), # Identifiers

('OP', r'[+\-\*/]'), # Arithmetic operators

('NEWLINE', r'\n'), # Line endings

('SKIP', r'[ \t]+'), # Skip over spaces and tabs

('MISMATCH', r'.'), # Any other character

]

tok\_regex = '|'.join('(?P<%s>%s)' % pair for pair in token\_specification)

line\_num = 1

line\_start = 0

for mo in re.finditer(tok\_regex, code):

kind = mo.lastgroup

value = mo.group()

column = mo.start() - line\_start

if kind == 'NUMBER':

value = float(value) if '.' in value else int(value)

elif kind == 'ID' and value in keywords:

kind = value

elif kind == 'NEWLINE':

line\_start = mo.end()

line\_num += 1

continue

elif kind == 'SKIP':

continue

elif kind == 'MISMATCH':

raise RuntimeError(f'{value!r} unexpected on line {line\_num}')

yield Token(kind, value, line\_num, column)

statements = '''

IF quantity THEN

total := total + price \* quantity;

tax := price \* 0.05;

ENDIF;

'''

for token in tokenize(statements):

print(token)