Testing und Debugging: Übung 04

Abgabetermin 26.11.2014 13:59

Name

Matrikelnummer

1 McCabe Code Metrik, Flussdiagramme und Co.

Neben der McCabe Metrik gibt es noch weitere Metriken, die etwas über die Qualität des Quellcodes aussagen.

Wichtig
Weitere Metriken:

1. Die Anzahl der Code-Zeilen (Abgekürzt mit *loc* für lines of code) pro Funktion.
2. Die Anzahl der Leerzeilen im Funktionscode (Abgekürzt mit *bl* für blank lines).
3. Anzahl der Kommentarzeilen (ohne inline Kommentare!) im Funktionscode (Abgekürzt mit *cl* für comment lines).

Konvention

Die 1. Zeile einer Funktion ist die Zeile mit der *def* Anweisung. Als letzte Zeile einer Funktion wird diejenige Zeile gewertet in der ein Python-Statement im Funktionsblock steht, dem kein weiteres Statement im Funktionsblock mehr folgt. Leerzeilen oder Kommentare am Ende zählen **nicht** mehr dazu.

Beispiel

```
1  # Funktion foo beginnt jetzt
2  def foo():
3      # 1. Kommentar
4
5      x = baz(42)
6
7  # 2. Kommentar
8      y = bar(1001)  # 3. Kommentar
9
10     # 4. Kommentar
11     return y, x
12
13     # Ende von foo
```

In obigem Beispiel gehören die Zeilen 2-11 zur Funktion und die Zeile 3-11 zum Funktionsblock. Damit gilt für *foo*:

- McCabe = 1
- loc = 11 - 2 + 1 = 10
- bl = 3 (Zeilen 4, 6 und 9. Nach obiger Konvention zählt Zeile 12 nicht mehr dazu.)
- cl = 3 (Kommentar 1, 2 und 4. Nach Definition zählt Kommentar 3 als inline Kommentar nicht dazu und der Kommentar aus Zeile 13 zählt laut Konvention nicht mehr dazu.)

Aufgabe (4+2+2+2+10 Punkte / Funktion)
Bestimmen Sie für alle Funktionen in den nachfolgenden Codeschnipseln jeweils die McCabe Metrik, loc, bl und cl. Zeichnen Sie außerdem zu jeder Funktion das Flußdiagramm.

1.1 Funktion01

```
class Uebung01:

    def Funktion03(self, x=10):
        assert isinstance(x, int)

        if x == 10:
            print("I'll do nothing else.")
            return

        while x != 0:
            x -= 1
            import time

            time.sleep(1)
```

1.2 Funktion02

```
def Funktion05(tol=0.001):
    dparams = [1.0, 1.0, 1.0]
    params = [0.0, 0.0, 0.0]
    best_error = run(params)
    n = 0
    while sum(dparams) > tol:
        for i in range(len(params)):
            params[i] += dparams[i]
            err = run(params)
            if err < best_error:
                best_error = err
                dparams[i] *= 1.1
            else:
                params[i] -= 2.0 * dparams[i]
                err = run(params)
                if err < best_error:
                    best_error = err
                    dparams[i] *= 1.1
                else:
                    params[i] += dparams[i]
                    dparams[i] *= 0.9
        n += 1
        print("Twiddle #", n, params, " -> ", best_error)
    return params
```

2 Programmieraufgabe: Unittests (200 Punkte)

Dateien für die Programmieraufgabe

Grading/*init*py	Hinweisdatei für Python: Das Verzeichnis ist ein Modul
Grading/Grading.py	Quellcode der Testklasse, die in *run_test.py* verwendet wird
uebung04.py *	In dieser Datei müssen Sie die Funktion *steuer* implementieren
uebung04_unit.py *	Implementieren Sie hier Ihre Unittests.

Die mit [*] markierten Dateien müssen **bearbeitet** und **abgegeben** werden.

2.1 Aufgabenstellung

Schreiben Sie eine Funktion, welches zu einem Brutto-Gehalt die zu zahlende Einkommenssteuer berechnet. Testen Sie Ihre Funktion mit Hilfe des Unittest Frameworks. Bei der Auswahl der Testfälle kann das Kriterium für Boundary Value Coverage hilfreich sein.

Gehalt in Euro	**Steuersatz**
bis einschl. 12.000	12 %
Über 12.000 bis einschl. 20.000	15 %
Über 20.000 bis einschl. 30.000	20 %
Über 30.000	25 %

Verheiratete Steuerpflichtigen werden 20% Ihrer Steuern erlassen, pro Kind werden 10% der Steuer erlassen (also bei drei Kindern 30%, nicht 27,1 %). Negative Steuersätze gibt es nicht.

Beispiel: 50000 Euro Jahresgehalt, verheiratet, ein Kind, ergibt eine Steuerlast von 50000*0.25*(1-(0.2+0.1)).

Die Funktion *steuer(bruttogehalt, verheiratet, kinder)* erwartet für *bruttogehalt* eine Fließkommazahl, und für *verheiratet* und *kinder* eine ganze Zahl. Für *verheiratet* gilt weiter, dass der Wert *1* für „ist verheiratet“ und *0* für „ist nicht verheiratet“ steht.

Der Rückgabewert der Funktion soll ein 2-Tupel (siehe Quellcode) der folgenden Form sein. In der ersten Komponente steht der berechnete Steuersatz und in der zweiten Komponente der Fehlercode:

- Bei einer nicht sinnvollen Eingabe wird *(0.0, 1)* zurück gegeben.
- Bei sinnvoller Eingabe wird *(steuersatz, 0)* zurück gegeben.

2.2 Bewertung

Ihr Programm wird zunächst gegen einen Satz von meinen Unittests getestet. Pro bestandenen Test wird es eine feste Anzahl von Punkten geben. Insgesamt können 100 Punkte erreicht werden.

Für jeden Prozentpunkt beim Branch Coverage **Ihrer** Unittests erhalten Sie je einen weitern Punkt.

2.3 Abgabe der Programmieraufgabe

1. Falls noch nicht vorhanden, dann erstellen Sie sich auch https://github.com einen Account. (Pro Gruppe ist nur ein Account und eine Abgabe nötig!)
2. Schicken Sie bei Ihrer ersten Abgabe den **Namen** des github Accounts (ohne Passwort) an sebastian.stigler@htw-aalen.de mit dem Betreff: *Testing und Debugging Github Account*
3. Gehen Sie in Ihrer virtuellen Maschine in das Aufgabenverzeichnis (wo sich Ihre bearbeitete Aufgabe und die Datei `submit.cfg` befinden).

4. Tippen Sie in der Konsole den Befehl *submit*. Dieser wird beim ersten Ausführen nach den Zugangsdaten Ihres github Accounts fragen. Anschließend werden die bearbeiteten Aufgaben verschlüsselt auf https://gist.github.com abgelegt.

5. Diese Datei wird nach dem Abgabetermin automatisch zur Korrektur heruntergeladen.

Viel Erfolg