Testing und Debugging: Übung 06

Abgabetermin siehe Canvas

1 Zufallsgesteuerte Testfallgeneratoren.

In dieser Übung wollen wir uns tiefer mit den zufallsgesteuerten generativen Testfallgeneratoren beschäftigen. Als praktisches Beispiel dient die Funktion *create_random_api_calls* aus der aktuellen Programmieraufgabe.

Diese Implementation soll die Vorteile zeigen, die eine geschickte Wahl von default Werten und die Möglichkeit komplexer Argumente für den Testfallgenerator bieten. Weiterhin soll es als Denkanstoß dienen, wenn Sie selbst einen solchen Generator entwickeln müssen.

Hilfreicher Link: http://docs.python.org/3.5/library/random.html.

1.1 Verstehen des internen Aufbaus und der Benutzung des Testfallgenerators (40 Punkte)

Gegeben sei folgender Auszug der gespeicherten Ausgabe von `create_random_api_calls()` (vollständige Datei liegt bei) die eine Fehler bei einer der fehlerhaften Implementationen der MyQueue Klasse auslöst.

API_test_stream

```
# api = 'empty', 'full', 'enqueue', 'dequeue'
# queues = 'MyQueue6'
# queue_size = None
# max_queue_size = 2000
# stream_length = None
# max_stream_length = 16000
# seed = 13681756363766
MyQueue6 1983
# chosen stream_length = 5267
empty
full
empty
dequeue
...
```

Aufgaben

1. *Warum wird für den Testfallgenerator kein echter Zufallsgenerator, sondern ein Pseudozufallsgenerator verwendet und welchen Vorteil können wir daraus ziehen? (5+5 Punkt)*

   .
   .
   .
   .

2. *Gegeben sei obige Ausgabe des Testfallgenerators. Kann man `create_random_api_calls()` so aufrufen, dass man* ***reproduzierbar*** *die exakt selbe Sequenz von API calls erzeugt bekommt, wie sie in der Datei* API_test_stream *gespeichert ist? Begründen Sie ihre Antwort. Geben Sie, wenn möglich, den Aufruf an, der die geforderte Sequenz erzeugt, oder begründen Sie weshalb dies nicht geht. (30 Punkte)*

.
.
.
.
.
.
.

1.2 Erst Denken, dann Ausprobieren (60 Punkte)

Ein erfolgreicher Tester haben muss sich von der gedanklichen Einstellung „Stocher ich mal wild drauf los und ruf dabei ein paar API-Funktionen auf; vielleicht stoße ich ja auf eine Bug.“ lösen.

Als Tester sollte man wie folgt vorgehen:

- Lesen und verstehen des Codes und der Spezifikation
- Identifizieren möglicher Schwachstellen im Code.
- Überlegen, wie die Software reagiert, wenn die Schwachstelle getroffen wurde; wie weicht das Ergebnis von der Spezifikation ab.
- Einen Test ausdenken und implementieren, der (hoffentlich) diese Schwachstelle triggert.
- Den Test laufenlassen und das Ergebnis beobachten und mit seinen Erwartungen vergleichen.
- Evtl. Test anpassen und erneut laufen lassen.

Aufgaben

1. *Wieso ist es sehr schwer, für **MyQueue3** mit `create_random_api_calls()` mit den default Einstellungen einen Test zu erzeugen, der den Fehler in MyQueue3 zu triggern? (20 Punkte)*

   .
   .
   .
   .
   .

2. *Geben Sie die kürzest mögliche Sequenz von API Aufrufen an, die den Fehler in MyQueue3 triggert. (Lange Sequenzen desselben API Aufrufs dürfen durch `Anzahl x API Call` abgekürzt werden.) (20 Punkte)*

   .
   .

3. *Geben Sie an, wie man `create_random_api_calls()` aufrufen muss um eine möglichst kurze Sequenz von API Aufrufen zu erhalten, die den Fehler **reproduzierbar** (d.h. bei jedem Aufruf von `create_random_api_calls()` mit ihren Parametern) auslöst. (20 Punkte)*

   .
   .
   .
   .

2 Programmieraufgabe: Random Testen der 8 fehlerhaften MyQueue Implementationen (vgl. Übung01). (200 Punkte)

Implementieren Sie einen Random API Tester für die 8 fehlerhaften MyQueues aus Übung01. Gehen Sie dabei gemäß der Graphik auf Folie 3 von Vorlesung05 vor. Verwenden Sie als Orakel den Vergleich mit einer Referenzimplementierung der MyQueue (vgl. Musterlösung der Programmieraufgabe von Übung01)

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
uebung06.py *	In dieser Datei müssen Sie die Funktionen `load_stream(filename)` und `run_stream(stream)` gemäß Aufgabenstellung implementieren.
MyQueue.py	In dieser Datei sind die 8 fehlerhaften Implementierungen der MyQueueklasse
test_MyQueue1-test_MyQueue8 †	Der Aufruf (evtl. mehrfach) von `python3 uebung06.py` füllt diese Dateien.

Die mit [*] markierten Dateien müssen **bearbeitet** und **abgegeben** werden.
Die mit [†] markierten Dateien sollten bei der Abgabe, von ‚create_random_api_calls()' erzeugte Sequenzen erhalten die den jeweiligen Fehler der angegebene MyQueue auslösen. Diese Dateien müssen mit **abgegeben** werden.

2.1 Aufgabenstellung

Implementieren Sie die Funktion `load_stream(filename)` so, dass nur ein gueltiger Datenstrom zurückgegeben wird (vgl. Kommentar im Skelett der Funktion in *uegung06.py*) eventuelle Leerzeilen in der Einzulesenden Datei werden beim Umwandeln in einen Datenstrom entfernt.

In der Funktion `run_stream(stream)` sollen die Funktionen aus der Variablen **API** automatisch getestet werden. Der Test eines API Aufrufs soll dabei wie folgt aussehen:

1. Aufrufen den eingelesenen API Aufrufs und speichern des Rückgabewertes.
2. Aufrufen des dazu passenden Aufrufs in der Referenzimplementation und speichern des Rückgabewertes.
3. Vergleichen der Rückgabewerte via Assertion.
4. Aufrufen von `check_rep()` für die zu testende MyQueue.

Via eines `try` Blocks werden die Assertions vom Vergleich und von `check_rep()` abgefangen.

Die Funktion führt nacheinander alle API Aufrufe wie eben beschrieben aus und bricht bei einem Fehler (API Rückgabe != Referenzrückgabe) sofort ab.

Die drei möglichen Rückgaben entnehmen Sie wieder dem Kommentar in der Funktion.

Die Funktionen `main_run()` und `load_all_run()` geben jeweils einen Punkt aus, wenn ein API Strom ohne Fehler durchläuft, ein **F** falls ein Fehler auftritt oder ein **X** falls ein ungültiger API Aufruf passiert ist.

Das **X** darf bei korrekter Implementation von `load_stream(stream)` und `run_stream(stream)` niemals ausgelöst werden!

2.2 Vorbereiten der Abgabe

Führen Sie das Programm mittels `python3 uebung06.py` aus, bis bei der Ausgabe von `load_all_run()` jede MyQueue ein **F** als Ausgabe erhalten hat.

Falls nach mehrfachem Aufruf die MyQueue3 immer noch kein **F** bekommt, dann Implementieren Sie die Funktion `main_run_Queue` wie im Kommentar am Ende von *uebung06.py* beschrieben.

Ziel ist es, dass Sie in den Dateien *test_MyQueue1 - test_MyQueue8* jeweils einen API Strom gespeichert haben, der den Fehler der jeweiligen MyQueue auslöst.

Schauen Sie sich diese Dateien vor der Abgabe an und vergewissern Sie sich, dass diese Dateien nicht leer oder nur aus Kommentarzeilen bestehen!!

2.3 Bewertung

Ihre 8 Ergebnisse werden gegen meine Musterlösung getestet. Diese Läufe müssen den jeweiligen Fehler auslösen und dürfen keine API Fehler enthalten.

Meine 8 Musterergebnisse werden in Ihre Funktionen geladen und auch diese müssen wie beschrieben reagieren.

Ihre Implementation wird gegen 4 fehlerhafte API Ströme getestet, die Entweder in der Lade Funktion abgefangen werden müssen oder aber in der `run_stream(stream)` Funktion mit dem Rückgabewert None quittiert werden müssen.

Für jede richtig behandelte Eingabedatei erhalten Sie 10 Punkte, also insgesamt 200 Punkte.

Eine Bewertung kann nur erfolgen, wenn beide Funktionen implementiert wurden!

Viel Erfolg